



## **ConnectPort X2** for Smart Energy

# **User's Manual**

**90001120\_F**  
SE\_Framework version 1.3.0.



---

©Digi International Inc. 2010. All Rights Reserved.

The Digi logo is a registered trademarks of Digi International, Inc.

Digi Connect, Connectware Manager, ConnectPort, Digi SureLink, are trademarks of Digi International, Inc.

All other trademarks mentioned in this document are the property of their respective owners.

Information in this document is subject to change without notice and does not represent a commitment on the part of Digi International.

Digi provides this document “as is,” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose. Digi may make improvements and/or changes in this manual or in the product(s) and/or the program(s) described in this manual at any time.

This product could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes may be incorporated in new editions of the publication.



---

# *Contents*

---

<b>Overview .....</b>	<b>1</b>
ZigBee Smart Energy .....	3
Description .....	3
Advantages .....	3
ConnectPort X2 for Smart Energy .....	4
Description .....	4
Network Diagram.....	6
Resources .....	7
Downloads.....	7
Online Documentation.....	7
 <b>Getting Started .....</b>	 <b>9</b>
Product Components, Requirements, And Resources .....	10
Product Components.....	10
PC Requirement.....	10
Additional Products .....	11
Set up iDigi and Gateway .....	12
Introduction to iDigi.....	12
Create an Account on iDigi.com.....	13
Configure the Gateway.....	15
Connect and Power on the ConnectPort X2 .....	15
Add Gateway to the idigi.com Device List .....	15
Add Devices to the ZigBee SE Network .....	18
Smart Energy Security Overview .....	18
Add Device to Trust Center Gateway .....	19
Joining a Router Gateway .....	19
Verify Device is on Gateway's Network.....	20
Get List of Devices on Network .....	20
Remove Device .....	22
iDigi SE Web Sample, Communicating with Gateway .....	23
Overview .....	23
How to Use this Sample .....	23
In-Premise Display/Meter Simulator Sample .....	29
Overview .....	29
Getting Started .....	29
Network View .....	34



In-Premise Display .....	35
Meter .....	39
<b>General Operation.....</b>	<b>40</b>
Startup Sequence .....	41
Remote Device Management .....	42
Device Detection .....	42
SE Client Clusters and Active Devices.....	42
SE Server Clusters and Active Devices .....	42
Transmission Retries and Device Inactivity .....	43
SE Server Clusters and Inactive Devices.....	43
Periodic Refresh .....	43
Explicit Device Add and Open Join .....	43
ZCL Reporting and Device Activity.....	44
Registry .....	45
Settings and Defaults .....	45
Power Safety .....	48
Global Saved Files .....	48
Hidden Saved Files .....	49
Basic Communication Overview .....	50
<b>Communication Protocol .....</b>	<b>50</b>
RPC Request and Response Example .....	51
Automatic Response Pushing .....	54
File Format .....	54
XML RPC Interface Overview.....	55
Conversion to Method Call .....	55
Parameter Type Specification Overview.....	56
Simple Parameter Types .....	57
Complex Parameter Types.....	58
Aliases.....	60
Request Identifier .....	61
Broadcasts .....	61
Synchronous Requests .....	61
XML RPC Interface Reference .....	63
General Requests/Responses .....	63
registry_configuration .....	63
<b>Appendix A .....</b>	<b>63</b>
add_module.....	64
remove_module.....	64
add_interface.....	65

remove_interface.....	65
add_endpoint.....	66
remove_endpoint.....	67
add_cluster.....	67
remove_cluster.....	68
get_version.....	69
help.....	69
exit.....	71
message (response only) .....	71
get_time.....	71
set_time.....	71
resync_time .....	72
ZigBee Requests/Responses .....	73
add_device .....	73
remove_device .....	73
enable_joining .....	74
refresh_device_information .....	74
get_device_information .....	74
xbee_AT .....	75
bind.....	75
unbind.....	76
configure_zigbee_network .....	77
leave_network .....	78
get_zigbee_network_status.....	78
ZCL Requests/Responses .....	79
read_attributes .....	79
write_attributes .....	80
discover_attributes .....	81
start_receiving_reports.....	82
attribute_report (response only) .....	83
stop_receiving_reports .....	84
stop_sending_reports.....	85
get_local_reporting_configurations .....	86
read_reporting_configuration .....	86
identify .....	88
send_ZCL .....	88
SE Requests/Responses .....	91
get_DRLC_events .....	91
create_DRLC_event.....	91
cancel_DRLC_event .....	92
cancel_all_DRLC_events .....	94
received_report_event_status (response only) .....	94
get_scheduled_DRLC_events.....	95
clear_DRLC_events .....	96
received_DRLC_event (response only) .....	97
received_DRLC_cancel_event (response only) .....	97
received_DRLC_cancel_all_events (response only).....	98

updated_active_DRLC_events (response only) .....	99
get_message_events .....	100
create_message_event .....	100
cancel_message_event .....	101
cancel_all_message_events .....	102
received_message_confirmation (response only) .....	103
received_message_event (response only) .....	104
received_message_cancel_event (response only) .....	104
updated_active_message_events (response only) .....	105
get_last_message_event .....	106
confirm_message .....	107
clear_message_events .....	108
get_price_events .....	109
create_price_event .....	109
cancel_all_price_events .....	110
get_current_price_event .....	112
get_scheduled_price_events .....	113
clear_price_events .....	113
received_price_event (response only) .....	114
Aliasing Commands .....	115
add_alias .....	115
remove_alias .....	115
list_aliases .....	116
Record Reference .....	117
ZDO Records .....	117
ZDODeviceRecord .....	117
Node_Desc_rsp .....	118
Power_Desc_rsp .....	119
Simple_Desc_rsp .....	119
ZCL Records .....	120
ReadAttributeRecord .....	120
ReadAttributeStatusRecord .....	120
WriteAttributeRecord .....	120
WriteAttributeResponseRecord .....	121
AttributeReportingConfigurationRecord .....	121
AttributeReportingConfigurationResponseRecord .....	122
AttributeReportRecord .....	122
StopReportingRecord .....	123
StopReportingStatusRecord .....	123
ReadReportingConfigurationRecord .....	123
ReadReportingConfigurationResponseRecord .....	123
AttributeInformationRecord .....	124
LocalReportingConfigurationRecord .....	125
ZCL_ArrayRecord .....	125
SE Records .....	126
LoadControlEventRecord .....	126

CancelLoadControlEventRecord .....	126
CancelAllLoadControlEventsRecord .....	127
ReportEventStatusRecord .....	127
GetScheduledEventsRecord .....	127
DisplayMessageRecord .....	128
CancelMessageRecord .....	128
MessageConfirmationRecord .....	129
GetScheduledEventsRecord .....	129
PublishPriceRecord .....	129
GetScheduledPricesRecord .....	130

## **Appendix B ..... 131**

Smart Energy Certificate Management.....	131
Certificates on ConnectPort X2 for Smart Energy .....	131
Certificates on Standalone XBee Modules .....	131
Obtaining Test Certificates .....	132
Determining EUI of a ConnectPort X2 for Smart Energy .....	132
Determining EUI of a Standalone XBee Module .....	132
Installing Certificates .....	133
Installing Certificates on the ConnectPort X2 for Smart Energy..	133
Installing Certificates on a Standalone XBee Module.....	134
Reverting/Uninstalling Certificates.....	134
Production Certificates and Modifications .....	134



---

# *Overview*

## **CONTENTS**

This document provides an introduction to Digi's Smart Energy Framework and explains how to use the framework to set up a ConnectPort X2™ running Python® code to implement a ZigBee® Smart Energy™ Gateway, how to use its provided samples, and details about its operation and XML RPC interface.

## **DROP-IN NETWORKING**

As an integral part of the Drop-in Networking strategy, the Python development environment is incorporated by Digi into each gateway. Digi's integration of the open Python scripting language provides customers a truly open standard for complete control over connections to devices, manipulation of data, and event-based actions. For more information about the Digi Python custom development platform, visit our Python portal today at:

<http://www.digi.com/technology/drop-in-networking/pdr.jsp>





## QUESTIONS?

For technical assistance with your Drop-in Network, call:

**1-800-903-8430 (US Only)**

Digi contact numbers outside US

Country	Toll Free Number
Argentina	00-800-3444-3666
Australia	0011-800-3444-3666
Brazil	0021-800-3444-3666
China North	00-800-3444-3666
China South	00-800-3444-3666
France	00-800-3444-3666
Germany	00-800-3444-3666
Hong Kong	001-800-3444-3666
India	000-800-100-3383
Israel	00-800-3444-3666
Italy	00-800-3444-3666
Japan	For calls from KDD fixed land-line phones: 010-800-3444-3666 From KDD public and mobile phones: 001-010-800-3444-3666 For non-KDD phones: 122-001-010-800-3444-3666
Korea	002-800-3444-3666
Mexico	001-800-903-8430
Netherlands	00-800-3444-3666
New Zealand	00-800-3444-3666
South Thailand	001-800-3444-3666
Spain	00-800-3444-3666
United Kingdom	00-800-3444-3666



## ZIGBEE SMART ENERGY

### Description

The ZigBee Smart Energy Profile defines a wireless home area network (HAN) to manage energy in residential areas. These networks are local to the home and connect through a gateway back to the Utility head-end application. The current devices defined for Smart Energy are:

- **Meter** - Reports consumption of energy, water, gas, etc.
- **Energy Service Interface (ESI)** - Gateway from the Utility head-end to the HAN. Formerly referred to as Energy Service Portal (ESP)
- **In-Premise Display (IPD)** - Displays consumption and pricing information for the consumer. Also referred to as In-Home Display (IHD)
- **Programmable Communicating Thermostat (PCT)** - Smart thermostat.
- **Load Control Device** - Can limit or turn off power to devices during high load times.
- **Range Extender** - Fills in gaps in wireless HAN.
- **Smart Appliance** (not currently defined)
- **Pre-Payment Terminal** (not currently defined)

### Advantages

Advantages of a ZigBee Smart Energy network:

- **Automated meter reading** - Remotely report meter usage for Electric, Gas, Water, etc. in real time.
- **Reduce peak power usage** - Accomplished through mandatory energy reduction events and financial motivation using pricing.
- **Empower customers** - Give customers access to real time meter usage data and pricing. Also creates a network connection to devices like thermostats in the home which may be used to remotely control set points and setbacks (not part of SE specification but possible through manufacturer extensions).



## CONNECTPORT X2 FOR SMART ENERGY

### Description

The Digi ConnectPort X2 for Smart Energy is a gateway on a Smart Energy network that provides secure access to a ZigBee Smart Energy network over the internet. The gateway is set up to take advantage of connection management services offered by the iDigi™ platform, and to intelligently handle SE events in order to reduce the need for communication and micro management by utility applications.

There are three Smart Energy certified software applications for the gateway on the ZigBee network:

Name	Smart Energy Device Type	Node Type
<b>ESI Coordinator</b>	Energy Service Interface (ESI)	Coordinator / Trust Center
<b>ESI Router</b>	Energy Service Interface (ESI)	Router
<b>Aux Gateway</b>	In-Premise Display (IPD)	Router

All of the configurations use the same core Python SE\_Framework codebase and share much of the same functionality. The difference between the node types are as follows:

- **Coordinator** - Forms the Smart Energy network and acts as the trust center. All devices joining the network must be authenticated with the trust center.
- **Router** - Joins the Smart Energy network through the trust center. Typically, a Smart Meter will act as the trust center on the network.

The Smart Energy device type determines whether Smart Energy clusters supported by the gateway are servers or clients. The ESI is defined by Smart Energy to support the server side clusters. These clusters create and cancel Smart Energy events; for example, publishing the price of commodities like electricity on the network. The Energy Service Interface supports the following clusters:

- **Time server** - Synchronizes real time clock for all of the devices on the ZigBee network.
- **Demand Response/Load Control (DRLC) server** - Manages DRLC events which can change temperature set points on thermostats, water heaters, and spas or turn off devices in the home to prevent brownouts during high peak energy situations.
- **Price server** - Manages pricing events that set the price of different commodities using a start time and duration in one minute intervals. For example, increasing the price of electricity during peak hours in the evening and decreasing price



during low demand times at night. Pricing can be set in real time to respond to demand or ahead of time to allow customers to plan energy usage.

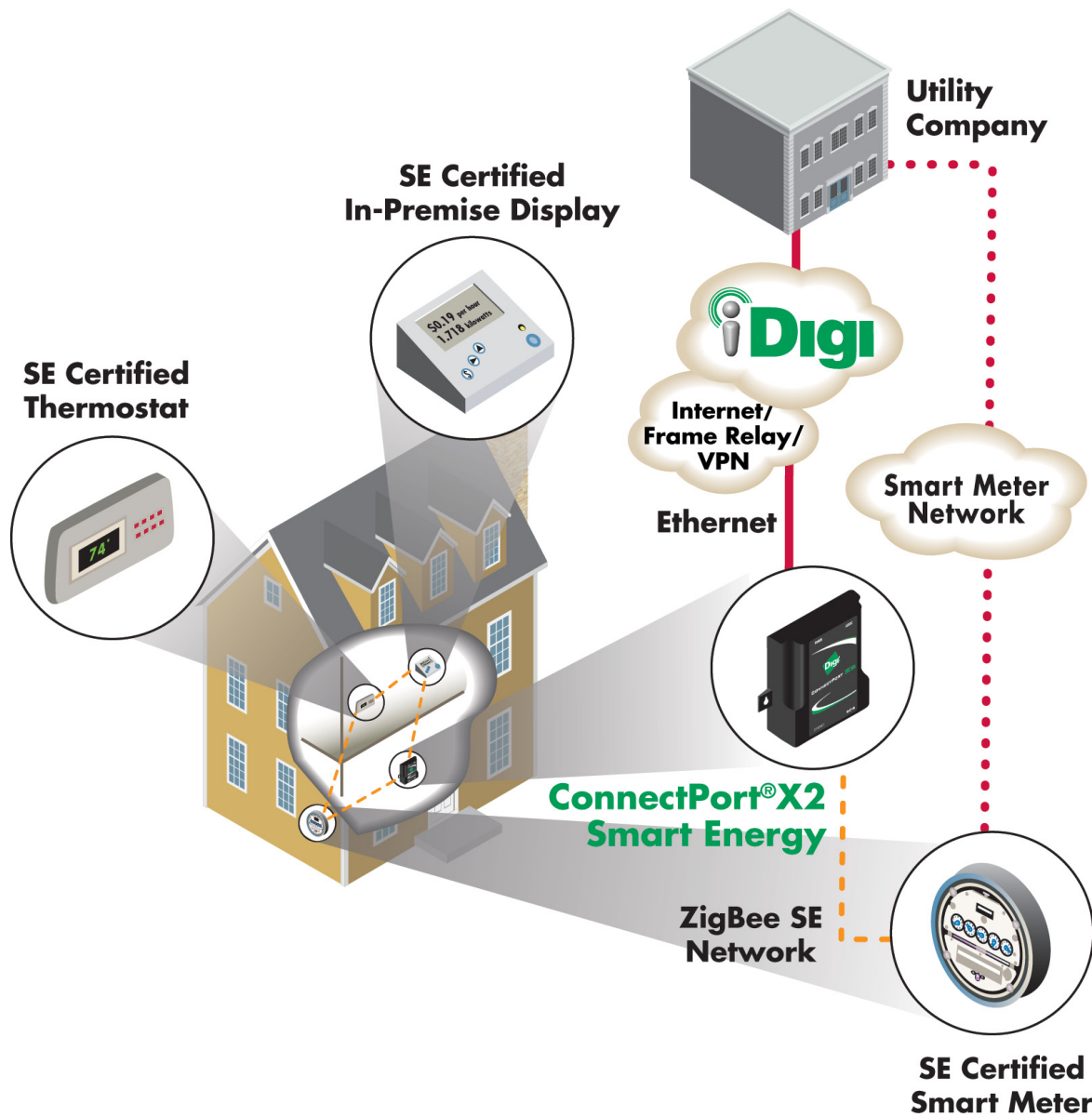
- **Messaging server** - Sends messages to a customer through ZigBee network. Messages can require confirmation from the customer.
- **Metering client** - Reads the metering usage data from meters on the network.

The client side clusters receive the events and act on them; for example, a thermostat may receive a published price event for electricity and change the heating setpoint to save money. The gateway makes these events available on the internet for other devices to act on, or for reporting purposes. The In-Premise Display Smart Energy device type supports the following clusters:

- **Time client** - Synchronizes real time clock with the time server on the ZigBee network.
- **Demand Response / Load Control (DRLC) client** - Receives DRLC events from the server and reports them through iDigi. The gateway handles canceled and overlapping events and provides updates when the currently active list of events changes.
- **Price client** - Receives price events for all available commodities from the price servers. The gateway also tracks when the prices of commodities change and provides updates through iDigi.
- **Messaging client** - Receives messages from the server and reports them through iDigi. The gateway allows users to confirm messages, handles canceled messages, and provides updates when the currently active message changes.
- **Metering client** - Reads the metering usage data from meters on the network.

## Network Diagram

This diagram shows the role of a Gateway within an Advanced Metering Infrastructure (AMI) network.





## RESOURCES

The following resources are referred to throughout this User's Manual. This page is intended to be a convenient reference.

The following download is available at [www.idigi.com](http://www.idigi.com).

- Python - version 2.4.3 (follow link to Python site)

## Downloads

Go to [www.digi.com/X2SE](http://www.digi.com/X2SE) or click on the link to download the following:

- Digi Device Discovery Application  
([http://ftp1.digi.com/support/utilities/40002265\\_G.exe](http://ftp1.digi.com/support/utilities/40002265_G.exe))
- ZB to Smart Energy (SE) Conversion Kit  
(<http://ftp1.digi.com/support/images/zb-to-smart-energy-conversion-kit.zip>)
- X-CTU  
(<http://www.digi.com/xctu>)

## Online Documentation

Go to [www.digi.com/X2SE](http://www.digi.com/X2SE) or click on the link to download the following on-line documents for detailed information on accessing and using the ConnectPort X2 for Smart Energy.

- ConnectPort X Series Documentation  
([http://ftp1.digi.com/support/documentation/90000832\\_a.pdf](http://ftp1.digi.com/support/documentation/90000832_a.pdf))
- XBee SE Manual  
<http://www.digi.com/products/wireless/zigbee-mesh/xbee-se-module.jsp>
- iDigi Web Services and Device Management Overview  
([http://developer.idigi.com/edocs/downloads/90001068\\_A.pdf](http://developer.idigi.com/edocs/downloads/90001068_A.pdf))

Note: You will first need an iDigi account to download.  
See "Create an Account on iDigi.com" on page 13.

## ZigBee Documentation

The referenced specifications are provided by the ZigBee Alliance and can be downloaded from their website at [www.zigbee.org](http://www.zigbee.org).

- ZigBee Smart Energy Test Specification, ZigBee Document 075384r17.
- ZigBee Cluster Library Specification, ZigBee Document 075123r02ZB.
- ZigBee Specification, ZigBee Document 053474r17.



- 
- ZigBee Smart Energy Profile Specification, ZigBee Document 075356r15.



## CHAPTER 1

# *Getting Started*

Upon completion of the Getting Started section you will be able to:

- Create an account on iDigi.com
- Configure your gateway
- Connect your gateway to iDigi.com
- Communicate with the gateway's RPC API through iDigi
- Add other devices to the gateway's network

The ConnectPort X2 for Smart Energy Starter Kit includes an XStick SE. You may also convert an XStick ZB, Digi XBee USB adapter, or other serially-attached XBee to SE. With one of these devices you will be able to:

- Simulate basic Smart Energy devices: In-Premise Display, Meter
- Add simulated devices to the gateway's network
- Communicate between the gateway and the simulated devices

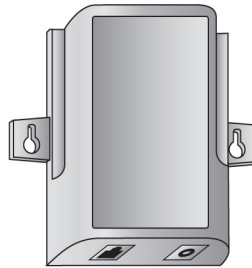
Devices on a Smart Energy network will vary widely between installations. Procedures in the User's Manual frequently specify generic tasks. See the manual that came with any devices you are joining to the gateway for details on how to apply these procedures. A sample is provided that uses a serially-attached XBee to simulate various Smart Energy devices and is useful for evaluation and development. When a procedure in this manual calls for a second device other than the gateway this sample can be used. (See "In-Premise Display/Meter Simulator Sample" on page 30.)

The user is also expected to have a means of sending RPC requests and receiving RPC responses from the gateway to access the Smart Energy Framework's API. In an actual installation this functionality would be provided by the utility application. In order to evaluate the gateway and for use during development a web application is provided to communicate with the gateway. When a procedure in this manual calls for sending an RPC request to your gateway this sample can be used. (See "iDigi SE Web Sample, Communicating with Gateway" on page 24.)



## PRODUCT COMPONENTS, REQUIREMENTS, AND RESOURCES

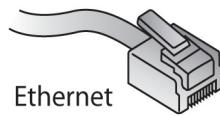
### Product Components



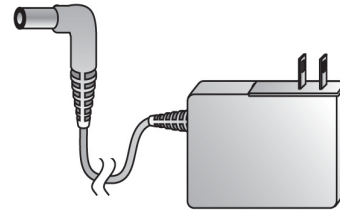
ConnectPort X2  
for Smart Energy



XStick for Smart Energy  
(only included in the Starter Kit)



Ethernet  
cable



Power supply

### PC Requirement

You will also need:



A personal computer,  
connected to the Internet.



## Additional Products

To run the In-Premise Display/Meter Simulator sample (see page 30), you will also need one of the following devices that has been converted to SE firmware. Full instructions and firmware can be found in the ZB to SE conversion kit, which can be downloaded from [www.digi.com/X2SE](http://www.digi.com/X2SE) or by clicking this link: <http://ftp1.digi.com/support/images/zb-to-smart-energy-conversion-kit.zip>.

- XBee-PRO ZB USB adapter with internal wire antenna (part number XA-Z14-CE1P-A). Available for purchase at <http://www.digi.com/products/wireless/zigbee-mesh/xbee-zb-adapters.jsp#models>.
- XStick USB Adapter, ZB (part number XU-Z11). Available for purchase at <http://www.digi.com/products/wireless/zigbee-mesh/xbee-zb-adapters.jsp#models>.
- XStick USB Adapter, SE (already configured with Smart Energy firmware (part number XU-SE3). Available for purchase at <http://www.digi.com/products/wireless/zigbee-mesh/xbee-zb-adapters.jsp#models>, included in the ConnectPort X2 for Smart Energy Starter Kit.
- Other serially-attached XBee if converted to SE firmware.

Note: The In-Premise Display/Meter Simulator sample only runs in Windows®



## SET UP iDIGI AND GATEWAY

### Introduction to iDigi

The iDigi Platform is a network management solution that provides easy integration for M2M (Machine-to-Machine) and mesh networking devices. iDigi is based on a cloud computing model that provides for on-demand scalability and high-availability. It lowers the barriers to building secure, scalable, cost-effective solutions that seamlessly tie together enterprise applications and remote devices regardless of their location or network.

iDigi is a secure, publicly accessible platform allowing remote devices to access it with little or no configuration. There is no need for connections to be opened through your DMZ in order for your devices to access iDigi. iDigi is accessed through the iDigi Portal, so by using a standard web browser you can configure network hardware, track device performance, remotely set filters and alarms, monitor device connections, device status and statistics, reboot devices, reset defaults, and remotely upgrade firmware for all the devices in your iDigi network.

See “Resources” on page 7 for a list of on-line documents that provide more information about iDigi.

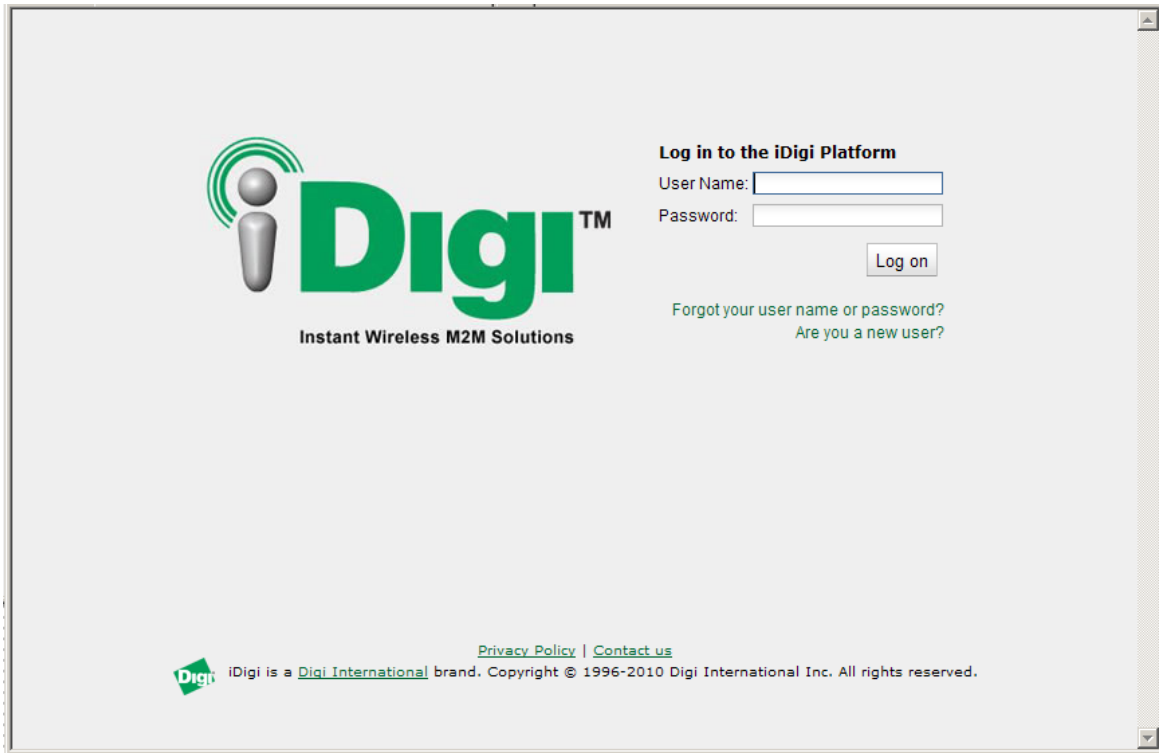
## Create an Account on iDigi.com

To get started, set up an account on the iDigi Platform as follows.

1. Navigate to <http://www.idigi.com>.
2. Click on the iDigi Platform Login button.



3. If you already have an account enter your user credentials in the User Name and Password fields, then click the Log on button. For new users, click on the "Are you a new user" link and create your account.



The screenshot shows the iDigi login interface. On the left is the iDigi logo with the tagline "Instant Wireless M2M Solutions". On the right, under the heading "Log in to the iDigi Platform", there are input fields for "User Name:" and "Password:". Below these is a "Log on" button. Further down, there are two links: "Forgot your user name or password?" and "Are you a new user?". At the bottom, there is a small Digi logo, a link to "Privacy Policy | Contact us", and a copyright notice: "iDigi is a Digi International brand. Copyright © 1996-2010 Digi International Inc. All rights reserved."



## Configure the Gateway

### Connect and Power on the ConnectPort X2

1. Unpack the ConnectPort X2 for Smart Energy gateway.
2. Connect the power supply to the X2 gateway and connect the power supply to an electrical outlet.

Note: (International version only): Connect the power supply to a power cable (not included), and the power cable to an outlet.

3. Connect an Ethernet cable from the gateway to your hub or switch that provides access to the Internet.

### Point Gateway to developer.idigi.com

By default, your Connectport X2 gateway points to energy.digi.com. This is our production server. For your development purposes, you can have up to five devices added to your account at developer.idigi.com.

Note: Older versions of the gateway may instead be pointed to device.digi.com. Use the same following steps to redirect the gateway to the correct server.

To configure your gateway properly, point the device to developer.idigi.com in the following manner:

- 1 Use the Digi Device Discovery application (see “Resources” on page 7) to determine your gateway’s IP address.
- 2 Telnet to this address, then type in the following commands:

```
login: root
```

```
password: dbps
```

```
#> set mgmtconnection svraddr1="http://developer.idigi.com" conntype=client
```

This will tell the gateway where to look for a server, specifically “http://developer.idigi.com. Now, type the “who” command to get the number of the process currently controlling where the gateway points. In this case, it’s process ID 1. Kill this process, as shown below.

```
#> who
```

<u>ID</u>	<u>From</u>	<u>To</u>	<u>Protocol</u>
1	10.8.16.83:45627	67.202.55.55:3197	idigi tcp
2			Python: main.py
3	10.8.16.85	local shell	telnet
4			Python thread



```
5 Python thread
6 Python thread
#>
#> kill 1
Connection 1 : killing connection ...
```

Now that the process is killed, you can type “who” again to see that the gateway is currently waiting (in this case, for ten seconds) to retry the server connection, which is now pointed to the correct server, developer.idigi.com.

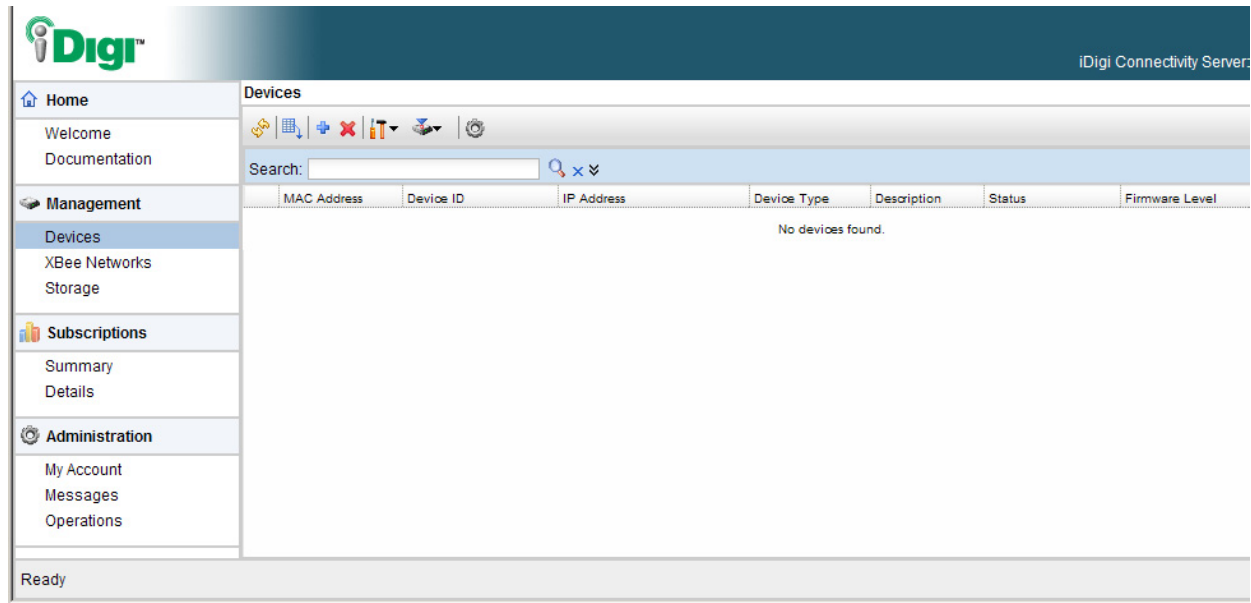
```
#> who
ID   From           To           Protocol
1   waiting to connect  wait 0:00:10  idigi wait
2                                     Python: main.py
3   10.8.16.85       local shell   telnet
4                                     Python thread
5                                     Python thread
6                                     Python thread
#>
```

After this process, you will be ready to add the gateway to your iDigi device list.

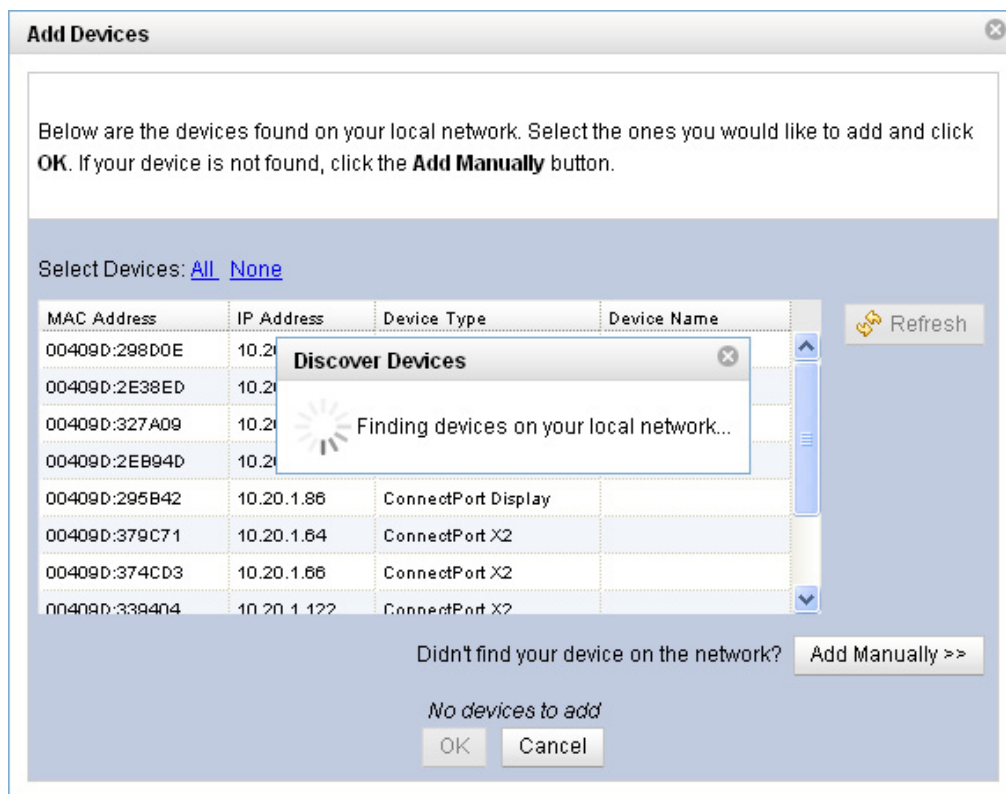
## Add Gateway to the idigi.com Device List

To add a gateway to the device list, follow these steps:

1. Log into the iDigi.com user portal using the username and password you just created. The iDigi Platform interface is displayed.



2. In the Devices list, click the  button to bring up the Add Devices dialog.

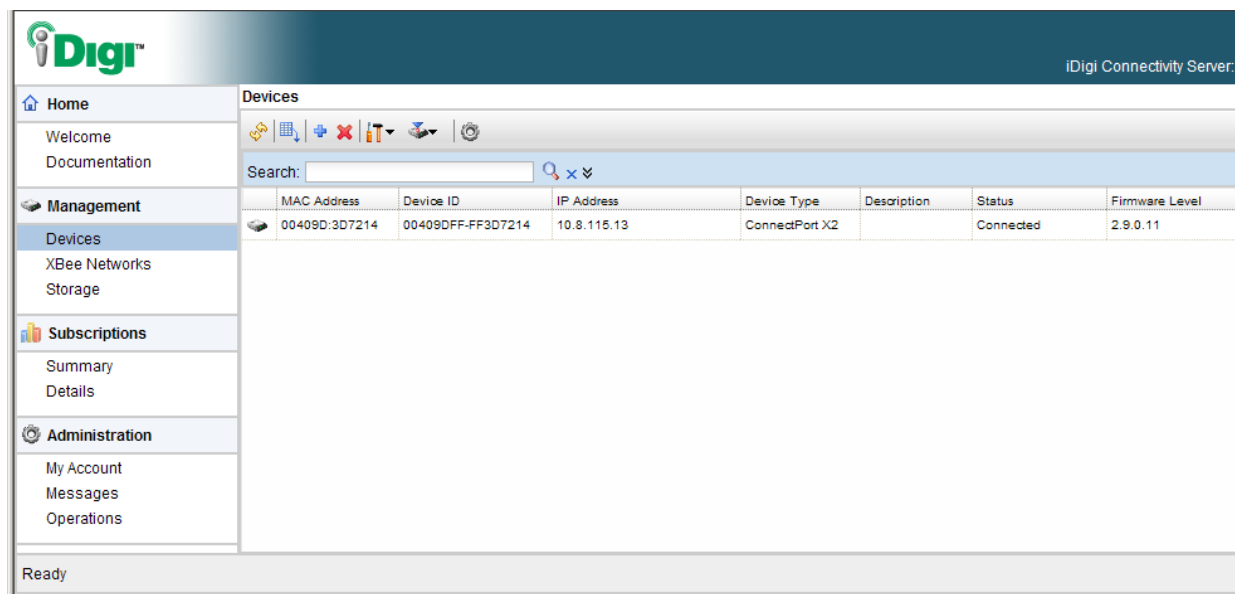




3. Locate and select your device from the list of locally discovered devices and click the 'OK' button. If your device was not found in the list, check that it is turned on and connected to the same local network as your PC and click the 'Refresh' button. Adding your device through automatic discovery informs iDigi about the device and configures that device to connect to the iDigi Connectivity server.

Note: If the device is not locally accessible or cannot be automatically discovered, you can still add it by clicking the 'Add Manually' button and enter the MAC address found on the bottom of the device.

4. Wait a few moments and click the **Refresh** button to ensure that your device status is now Connected.



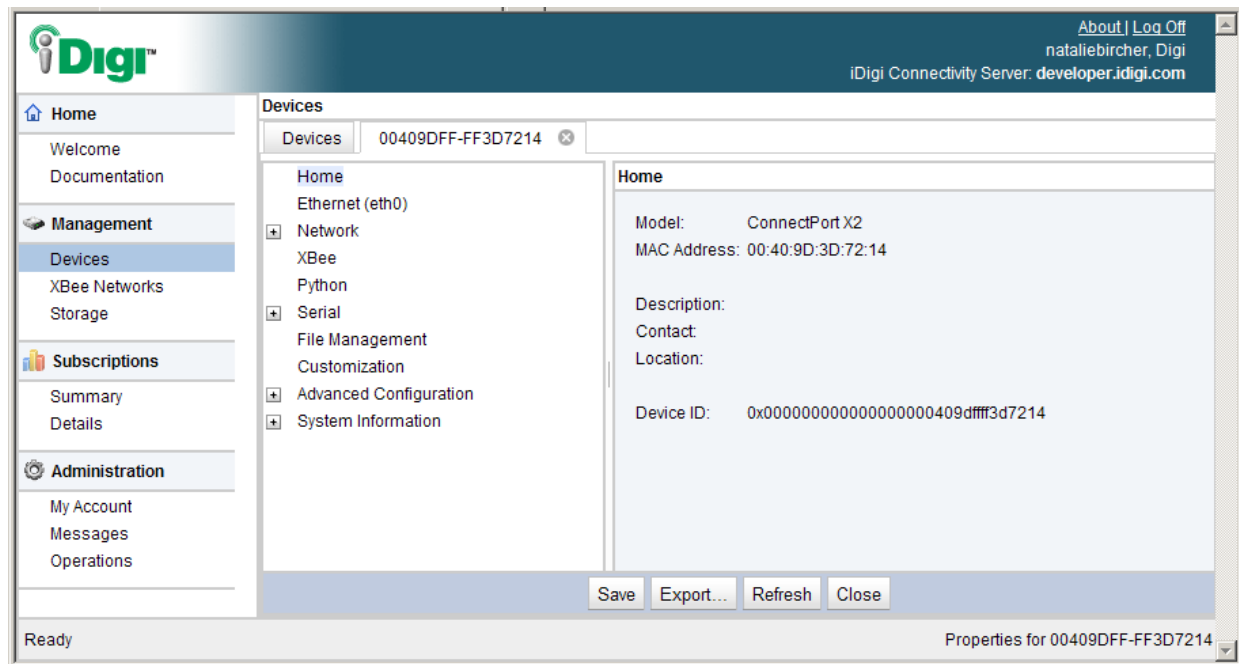
The screenshot shows the iDigi Connectivity Server web interface. On the left is a navigation menu with sections: Home (Welcome, Documentation), Management (Devices, XBee Networks, Storage), Subscriptions (Summary, Details), and Administration (My Account, Messages, Operations). The 'Devices' section is currently selected. The main area displays a table of discovered devices. Above the table is a search bar and several action icons. The table has columns for MAC Address, Device ID, IP Address, Device Type, Description, Status, and Firmware Level. One device is listed with the following details:

MAC Address	Device ID	IP Address	Device Type	Description	Status	Firmware Level
00409D:3D7214	00409DFF-FF3D7214	10.8.115.13	ConnectPort X2		Connected	2.9.0.11

At the bottom of the interface, a status bar indicates 'Ready'.

5. Select your device and double-click it, or right-click and select Properties.

6. Your device information will load into a separate tab.





## ADD DEVICES TO THE ZIGBEE SE NETWORK

The Google App sample may be used to send RPC requests to the gateway. (See “iDigi SE Web Sample, Communicating with Gateway” on page 24.) Please read this section first if you do not have a means of sending RPC requests to the gateway. Additionally, the In-Premise Display/Meter Simulator sample may be used to simulate devices to add to your network. (See “In-Premise Display/Meter Simulator Sample” on page 30.)

### Smart Energy Security Overview

Smart Energy imposes security protocols above and beyond a normal ZigBee network. Devices join the encrypted Smart Energy network using a pre-shared link key or installation code (installation codes are hashed into link keys). In a Smart Energy network, each device can have its own link key or installation code. After a device joins the network it will initiate key establishment.

During key establishment the trust center will establish an APS link key with the device. This key is used to encrypt critical data between the trust center and the joining device so that no other device can decrypt the data portion of the message. The trust center can also establish APS keys for any two authenticated devices so that they can securely communicate as well.

Key establishment utilizes certificates on both the trust center and joining device to authenticate that the device has been certified. Generally certificates are either production certificates or test certificates and all devices on a network must be of the same type. The X2 gateway is manufactured with a production certificate. See “Appendix B” on page 131 for instructions on obtaining and installing a test certificate.

The following generic steps need to be taken for a device to join:

1. Install test or production certificate on joining device to match Smart Energy network. If joining device is already configured with an appropriate certificate this step may not be necessary.
2. Register link key/installation code of joining device with the trust center.
3. Enable joining on the Smart Energy network.
4. Instruct joining device to join the Smart Energy network.



## Add Device to Trust Center Gateway

In order to add a device to the Smart Energy network with the X2 gateway running as a trust center (ESI coordinator), use the `add_device` RPC request (see “`add_device_response` Parameters:” on page 73). This will add a device with the given link key or installation code to the device table stored in the ESI. Joining will be enabled for the specified amount of time (set in this example to 600 seconds). Fill in the `device_address` and `link_key` as appropriate to match the joining device. If using an installation code replace `link_key` with `installation_code`.

This request has the following format:

```
<add_device>
  <device_address type="MAC">11:22:33:44:55:66:77:88</device_address>
  <join_time>600</join_time>
  <link_key type="base16">56777777777777777777777777777777</link_key>
</add_device>
```

Once the `add_device` RPC request has been sent, instruct the joining device to join the network. Once a device joins the network, the gateway will return a message with the following format:

```
<message timestamp="1273104009.18">
  <description type="string">
    ZDO_Device_Manager - Device 11:22:33:44:55:66:77:88 detected and marked as
    active
  </description>
<severity type="int">0x1</severity>
</message>
```

## Joining a Router Gateway

By default, router gateways (ESI router and Aux Gateway) will automatically try to join any network on any channel on startup using their pre-configured installation code. These joining parameters can be configured with the `get_zigbee_network_configuration` RPC request (see “`get_zigbee_network_configuration`” on page 78). This request configures the interval between join attempts, the scan channel mask for joining, the installation code and the extended PAN ID.

```
<get_zigbee_network_configuration>
  <extended_pan_id>0</extended_pan_id>
  <installation_code type="base16">1234567890ABBA9E</installation_code>
  <join_attempt_interval>120</join_attempt_interval>
  <channel_mask>0xFFFF</channel_mask>
  <eui type="MAC">00:11:22:33:44:55:66:77</eui>
</get_zigbee_network_configuration>
```

All of the parameters are optional. The response will give the current value for everything but the link key and the installation code. Once a router gateway has joined a network, the gateway will return a message with the following format:

```
<message>
  <status type="int">0x1</status>
  <description type="string">Local device detected that it is joined to a ZigBee
network</description>
  <severity type="int">0x1</severity>
</message>
```

## Verify Device is on Gateway's Network

To verify that a device is joined to the gateway's network, use the identify RPC request (see "identify" on page 88). This request should make the device physically identify itself in some way. The X2 gateways will blink the association LED at a faster rate after receiving a ZCL Identify command.

This request has the following format:

```
<identify>
  <destination_address type="MAC">11:22:33:44:55:66:77:88</destination_address>
  <source_endpoint_id>0x5E</source_endpoint_id>
  <destination_endpoint_id>0xFF</destination_endpoint_id>
  <identify_time>30</identify_time>
</identify>
```

Fill in the `destination_address` and `destination_endpoint_id` as appropriate to match the joining device. The `identify_time` parameter specifies how long the device should identify itself in seconds.

## Get List of Devices on Network

To get a list of devices on a gateway's network, use the `get_device_information` RPC request (see "get\_device\_information" on page 74). This request will retrieve all currently known information about all devices on the network from the gateway, including endpoints and clusters. The response will show which devices are active - have joined the network and are currently communicating with the gateway.

This RPC request has the following format:

```
< get_device_information />
```

The response has the following format:



```
<get_device_information_response timestamp="1257799818.0">
  <record_list type="list">
```

### This is our local device

```
    <item type="ZDODeviceRecord">
      <active_endpoints type="list">
        <item>0x5E</item>
      </active_endpoints>
      <power_descriptor type="Power_Desc_rsp">
        (See "Power_Desc_rsp" on page 118 for parameters.)
      </power_descriptor>
      <node_type>0x0</node_type>
      <addr_short type="MAC">0000</addr_short>
      <addr_extended type="MAC">00:13:A2:00:40:5C:01:F7</addr_extended>
      <node_descriptor type="Node_Desc_rsp">
        (See "Node_Desc_rsp" on page 117 for parameters.)
      </node_descriptor>
      <active type="bool">TRUE</active>
      <manufacturer_id>0x101E</manufacturer_id>
      <simple_descriptors type="dict">
        <endpoint_0x5E type="Simple_Desc_rsp">
          (See "Simple_Desc_rsp" on page 118 for parameters.)
        </endpoint_0x5E>
      </simple_descriptors>
    </item>
```

### This is a device that has not joined yet

```
    <item index="2" type="ZDODeviceRecord">
      <active type="bool">FALSE</active>
      <addr_extended type="MAC">
        11:22:33:44:55:66:77:88
      </addr_extended>
      <node_type>-0x1</node_type>
    </item>
```

### This is a device that has joined and is currently active

```
    <item index="3" type="ZDODeviceRecord">
```



```
<active_endpoints type="list">
  <item>0x5E</item>
</active_endpoints>
<power_descriptor type="Power_Desc_rsp">
  (See "Power_Desc_rsp" on page 118 for parameters.)
</power_descriptor>
<node_type>0x1</node_type>
<addr_short type="MAC">1397</addr_short>
<addr_extended type="MAC">77:66:55:44:33:22:11:00</addr_extended>
<node_descriptor type="Node_Desc_rsp">
  (See "Node_Desc_rsp" on page 117 for parameters.)
</node_descriptor>
<active type="bool">TRUE</active>
<manufacturer_id>0x101E</manufacturer_id>
<simple_descriptors type="dict">
  <endpoint_0x5E type="Simple_Desc_rsp">
    (See "Simple_Desc_rsp" on page 118 for parameters.)
  </endpoint_0x5E>
</simple_descriptors>
</item>
</record_list>
</get_device_information_response>
```

## Remove Device

To remove a device from the network, use the `remove_device` RPC request (see “`remove_device`” on page 73). This request will send a ZDO `Mgmt_Leave_req` to the device, remove the device from the list returned by `get_device_information_response`, and if the gateway is a trust center (ESI coordinator), unregister the link key / installation code for this device from the XBee.

This request has the following format:

```
<remove_device>
  <device_address type="MAC">
    11:22:33:44:55:66:77:88
  </device_address>
</remove_device>
```

## iDIGI SE WEB SAMPLE, COMMUNICATING WITH GATEWAY

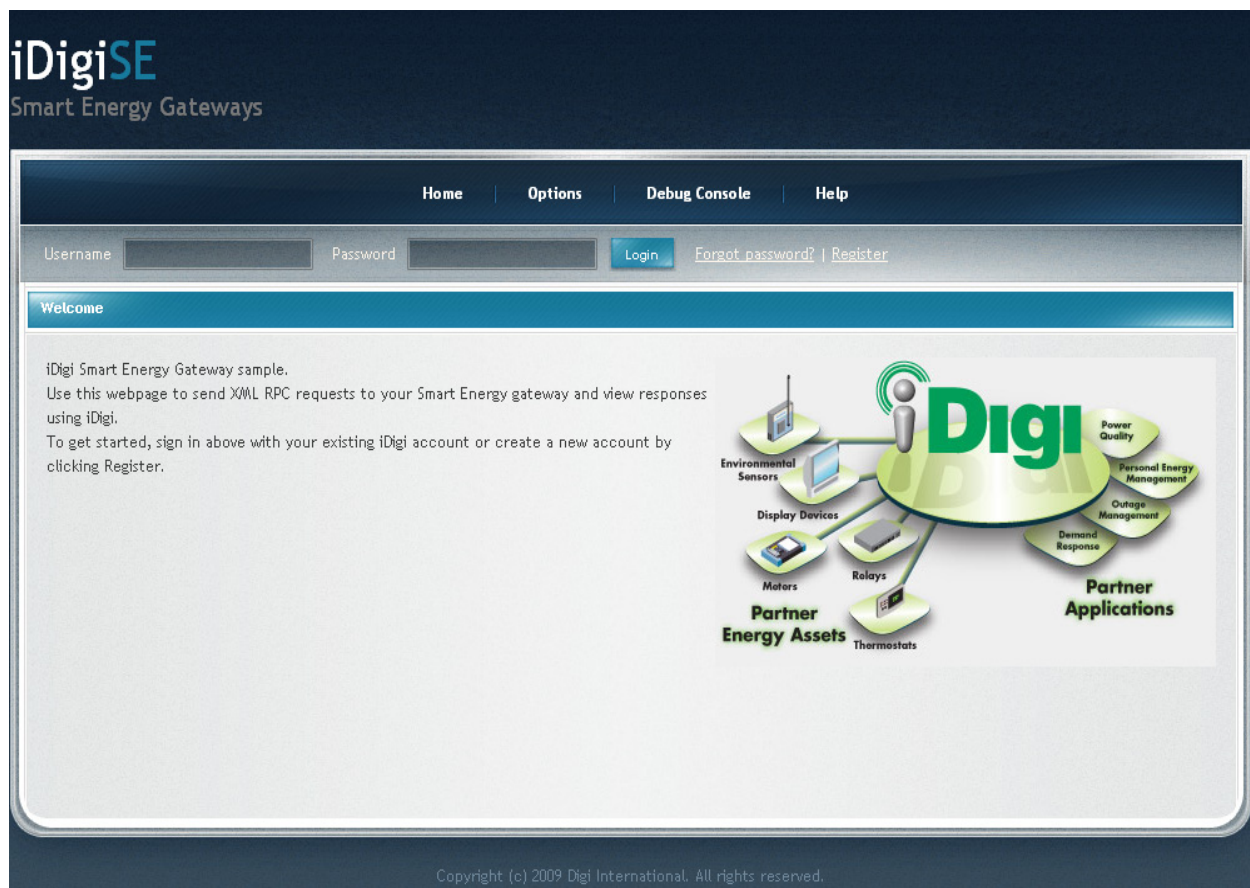
### Overview

This sample provides a simple demonstration of a system which communicates with a ConnectPort X2 for Smart Energy through iDigi using the RPC interface.

Because this sample provides generic access to the RPC interface, it may be used to communicate with the gateway for other samples and development.

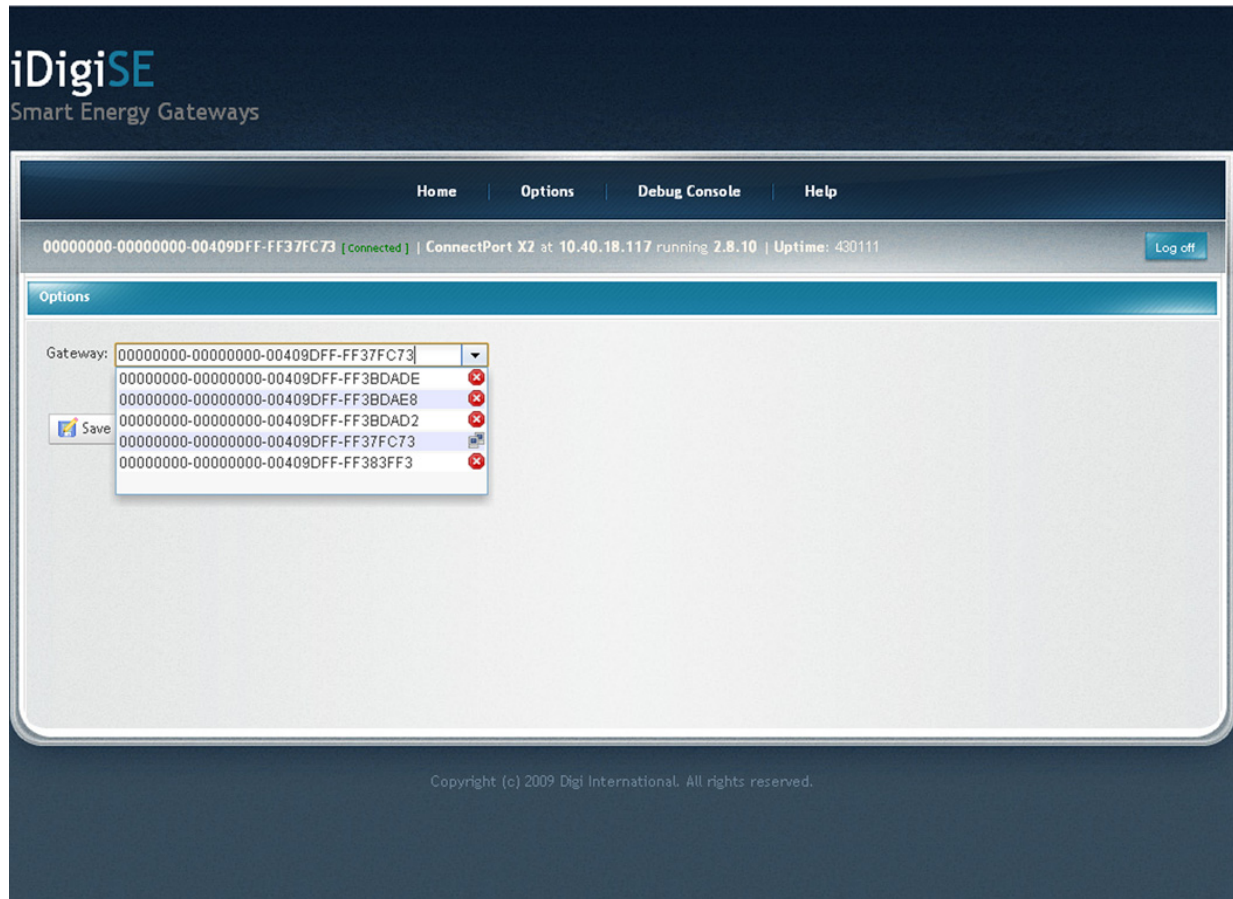
### How to Use this Sample

1. Go to [www.idigi-se.appspot.com](http://www.idigi-se.appspot.com).
2. Enter your iDigi username and password and click **Login**.

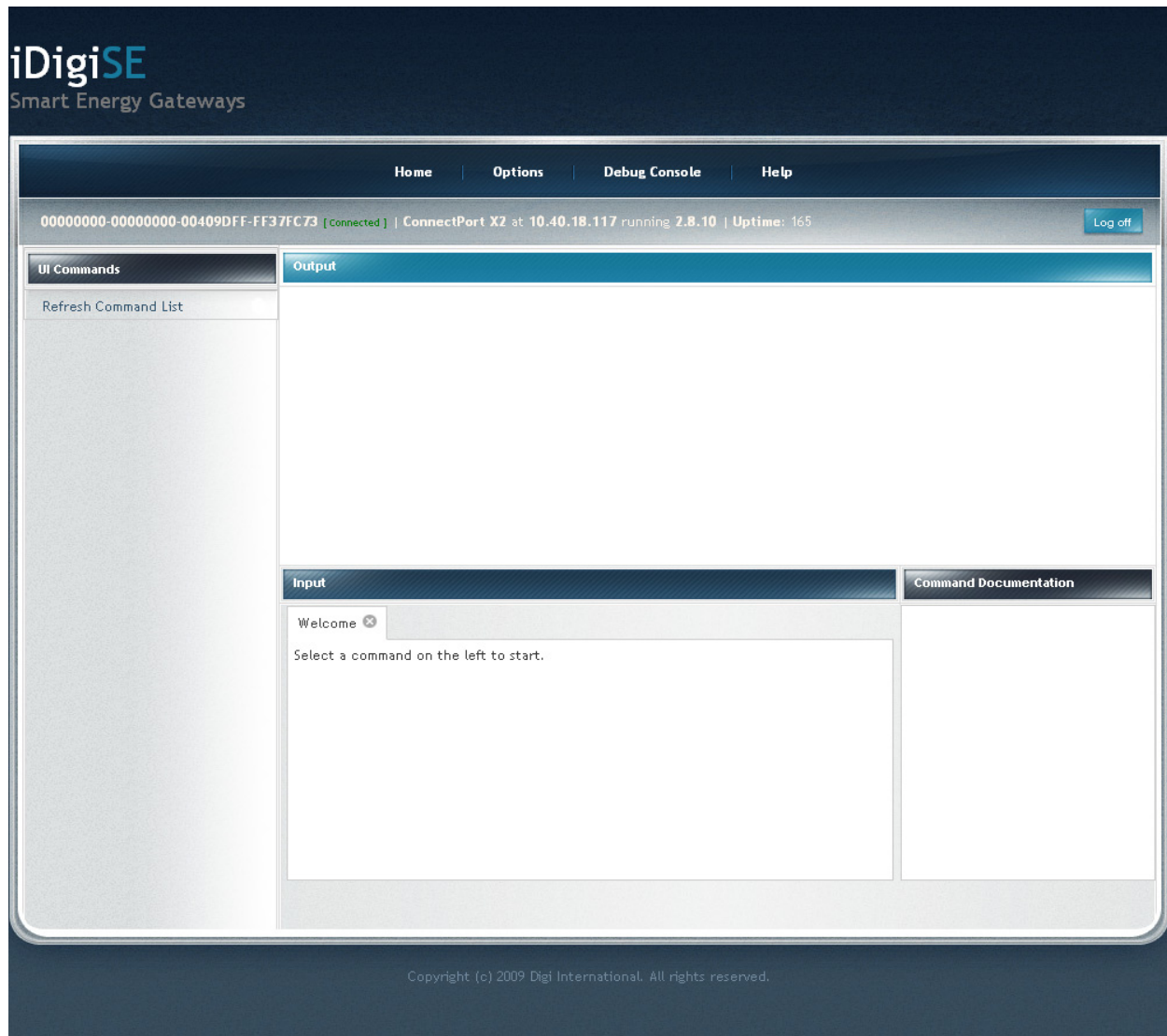




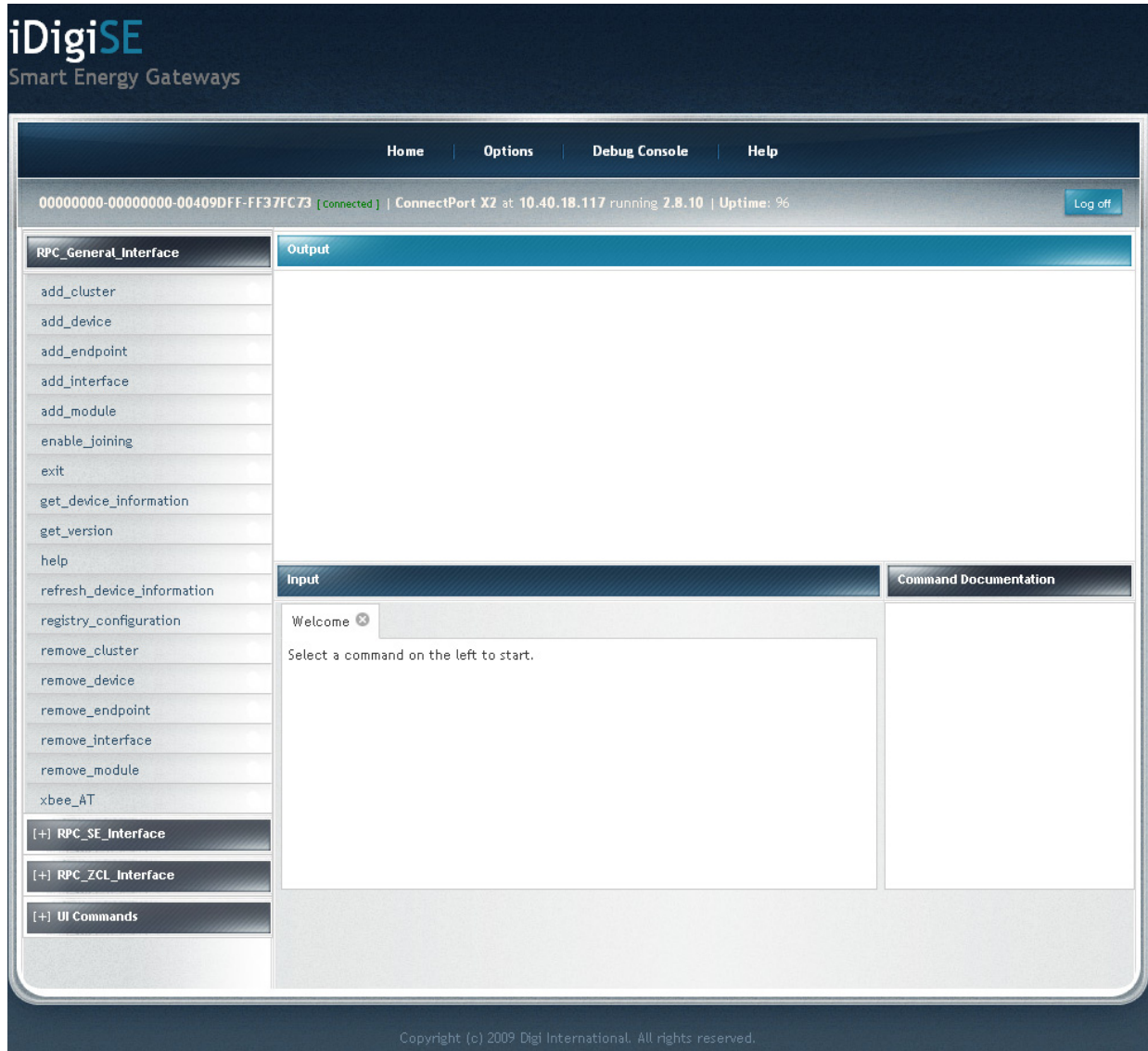
3. On the Options window, select the device ID of your gateway and click **Save**.



4. On the Debug Console page, messages sent by the gateway will appear in the output window. If the gateway has just been turned on, some initialization messages may be displayed in the window.



5. Query the gateway for available commands by clicking the **Refresh Command List** button. This will populate the command list.



**iDigiSE**  
Smart Energy Gateways

Home | Options | Debug Console | Help

00000000-00000000-00409DFF-FF37FC73 [connected] | ConnectPort X2 at 10.40.18.117 running 2.8.10 | Uptime: 96 [Log off](#)

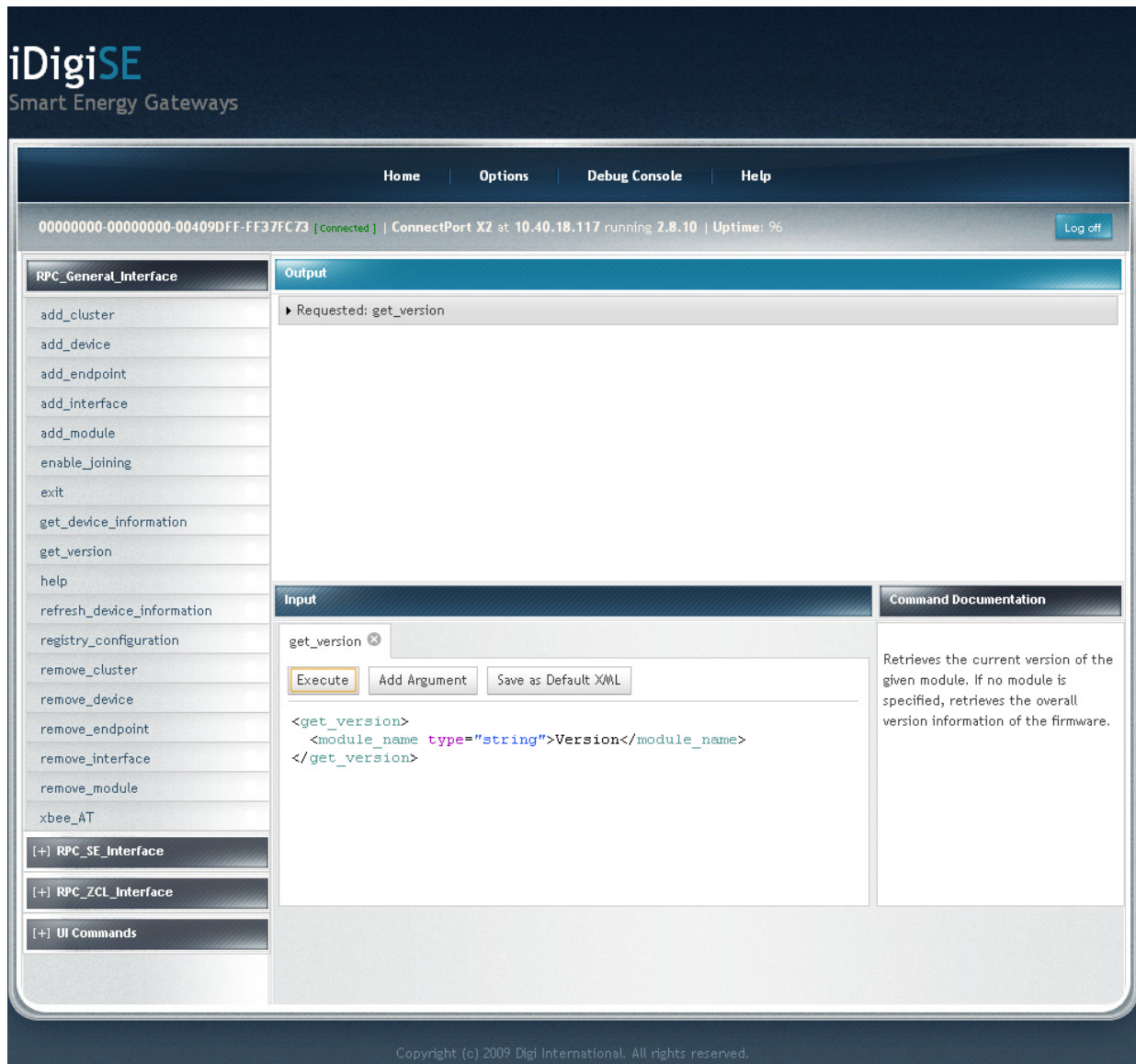
RPC_General_Interface	Output
<ul style="list-style-type: none"> <li>add_cluster</li> <li>add_device</li> <li>add_endpoint</li> <li>add_interface</li> <li>add_module</li> <li>enable_joining</li> <li>exit</li> <li>get_device_information</li> <li>get_version</li> <li>help</li> <li>refresh_device_information</li> <li>registry_configuration</li> <li>remove_cluster</li> <li>remove_device</li> <li>remove_endpoint</li> <li>remove_interface</li> <li>remove_module</li> <li>xbee_AT</li> </ul>	

Input	Command Documentation
<p>Welcome ✕</p> <p>Select a command on the left to start.</p>	

[\[+\] RPC\\_SE\\_Interface](#)  
[\[+\] RPC\\_ZCL\\_Interface](#)  
[\[+\] UI Commands](#)

Copyright (c) 2009 Digi International. All rights reserved.

6. Select a command from the list and modify parameters as necessary. Click **Execute**.



The screenshot shows the iDigiSE web interface. At the top, there's a navigation bar with 'Home', 'Options', 'Debug Console', and 'Help'. Below this, a status bar displays the device ID '00000000-00000000-00409DFF-FF37FC73' as 'connected', the connection details 'ConnectPort X2 at 10.40.18.117 running 2.8.10', and the 'Uptime: 96'. A 'Log off' button is on the right.

The main interface is divided into three sections:

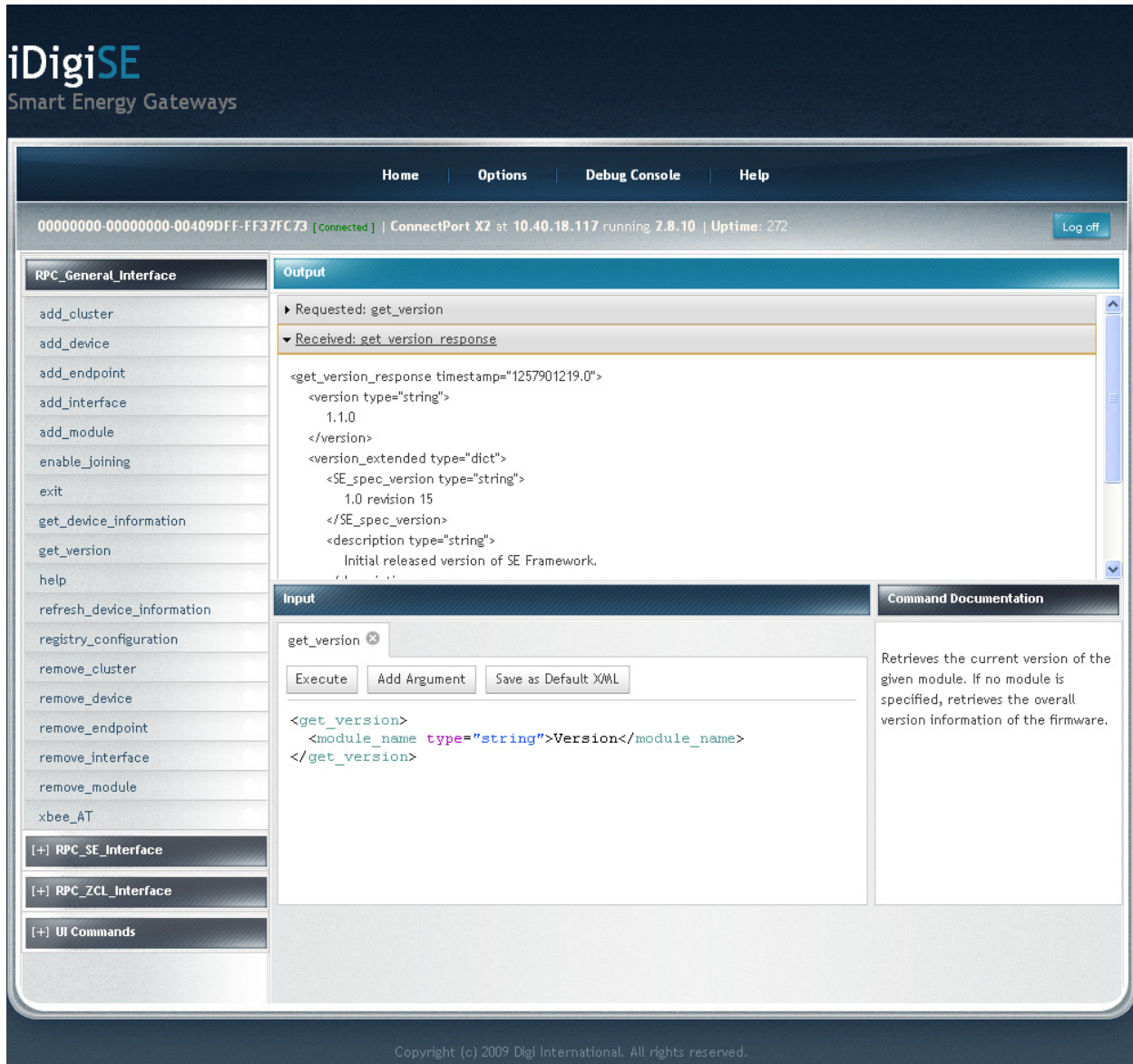
- Left Panel (RPC\_General\_Interface):** A list of commands including 'add\_cluster', 'add\_device', 'add\_endpoint', 'add\_interface', 'add\_module', 'enable\_joining', 'exit', 'get\_device\_information', 'get\_version', 'help', 'refresh\_device\_information', 'registry\_configuration', 'remove\_cluster', 'remove\_device', 'remove\_endpoint', 'remove\_interface', 'remove\_module', 'xbee\_AT', and expandable sections for 'RPC\_SE\_Interface', 'RPC\_ZCL\_Interface', and 'UI Commands'.
- Output:** A section showing the result of the executed command, displaying 'Requested: get\_version'.
- Input:** A section for configuring the command execution. It shows 'get\_version' with an 'X' icon. Below it are buttons for 'Execute', 'Add Argument', and 'Save as Default XML'. The XML input field contains:
 

```
<get_version>
  <module_name type="string">Version</module_name>
</get_version>
```
- Command Documentation:** A section providing details for the 'get\_version' command: 'Retrieves the current version of the given module. If no module is specified, retrieves the overall version information of the firmware.'

At the bottom of the interface, a copyright notice reads: 'Copyright (c) 2009 Digi International. All rights reserved.'



XML requests and responses will appear in the output window and can be expanded by clicking on them.



The screenshot displays the iDigiSE Smart Energy Gateways web interface. The top navigation bar includes links for Home, Options, Debug Console, and Help. A status bar at the top indicates the device is connected (00000000-00000000-00409DFF-FF37FC73) and provides details about the ConnectPort X2 at 10.40.18.117, running version 2.8.10, with an uptime of 272. A Log off button is also present.

The main interface is divided into several sections:

- RPC\_General\_Interface:** A list of commands including add\_cluster, add\_device, add\_endpoint, add\_interface, add\_module, enable\_joining, exit, get\_device\_information, get\_version, help, refresh\_device\_information, registry\_configuration, remove\_cluster, remove\_device, remove\_endpoint, remove\_interface, remove\_module, xbee\_AT, and expandable sections for RPC\_SE\_Interface, RPC\_ZCL\_Interface, and UI Commands.
- Output:** A window showing the execution of the get\_version command. It displays a "Requested: get\_version" and a "Received: get\_version response" containing XML data:
 

```
<get_version_response timestamp="1257901219.0">
  <version type="string">
    1.1.0
  </version>
  <version_extended type="dict">
    <SE_spec_version type="string">
      1.0 revision 15
    </SE_spec_version>
    <description type="string">
      Initial released version of SE Framework.
    </description>
  </version_extended>
</get_version_response>
```
- Input:** A section for entering commands. It shows the "get\_version" command selected, with buttons for "Execute", "Add Argument", and "Save as Default XML". The XML input field contains:
 

```
<get_version>
  <module_name type="string">Version</module_name>
</get_version>
```
- Command Documentation:** A section providing details for the selected command. For "get\_version", it states: "Retrieves the current version of the given module. If no module is specified, retrieves the overall version information of the firmware."

The footer of the interface includes the copyright notice: "Copyright (c) 2009 Digi International. All rights reserved."



## IN-PREMISE DISPLAY/METER SIMULATOR SAMPLE

### Overview

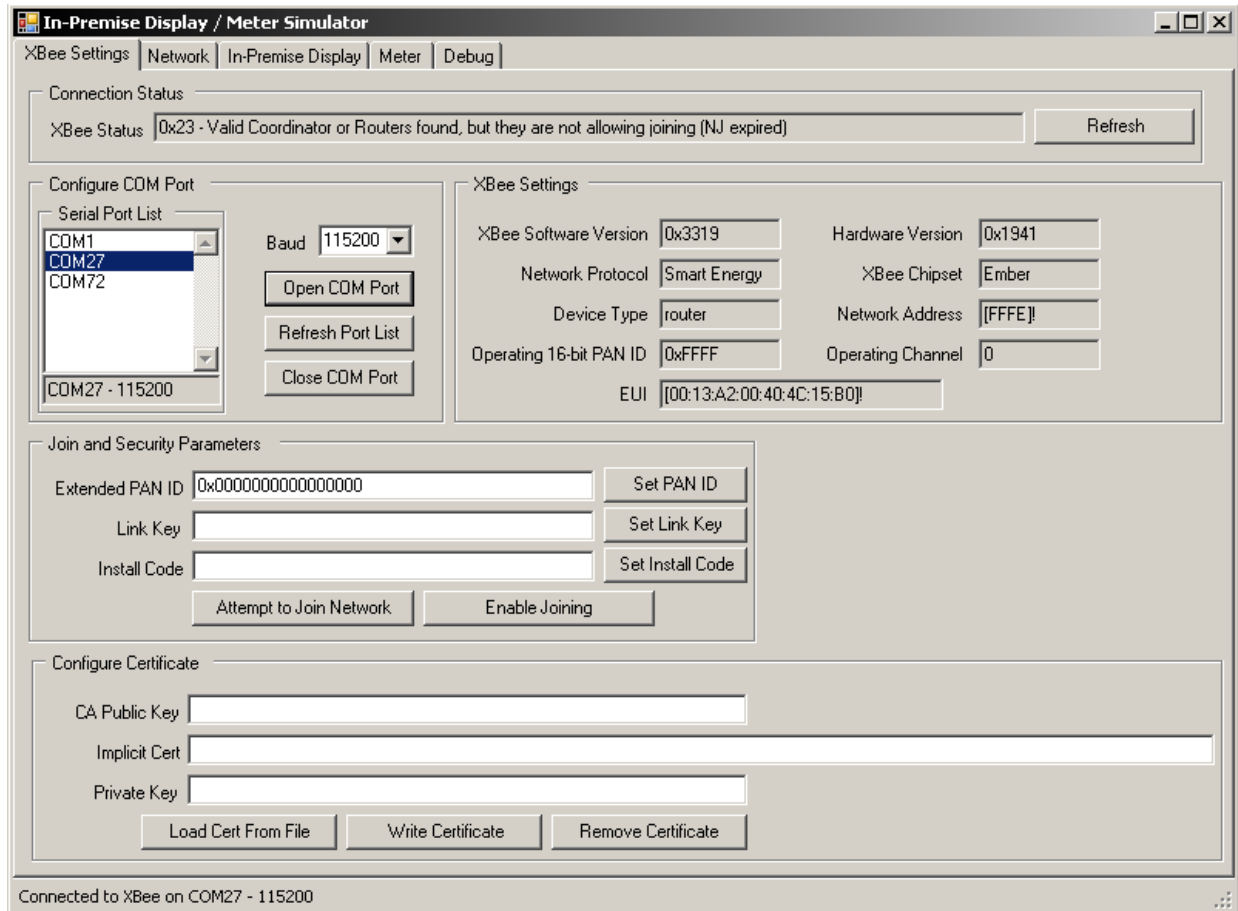
This sample simulates the basic functionality of a Smart Energy In-Premise Display or Meter with an XBee attached to a personal computer such as an XStick, XBee USB adapter, or other serially-attached XBee. It can be used to demonstrate Smart Energy functionality when real SE IPDs and meters are not convenient or available. If you do not already have one of these devices they may be purchased from the Digi website.

XBee modules are not Smart Energy certified for standalone use. In order to join the XBee module to the gateway a test certificate will need to be installed on both the XBee module and the gateway. See "Appendix B" on page 131 for more details on installing test certificates. In particular, obtain a test certificate for both the gateway and the XBee module and install the gateway's test certificate prior to executing the following procedure.

### Getting Started

1. Connect the serially-attached XBee to your personal computer.
2. Upgrade the serially-attached XBee to use Smart Energy router firmware if it is not already. Full instructions and firmware can be found in the ZB to SE conversion kit at <http://ftp1.digi.com/support/images/zb-to-smart-energy-conversion-kit.zip>.
3. Download and install the In-Premise Display/Meter Simulator from [www.digi.com/X2SE](http://www.digi.com/X2SE)
4. Start In-Premise Display/Meter Simulator.

5. Select the COM port and baud rate of your serially-attached XBee and click **Open COM Port**. The Baud rate is shown below as 115200 but will commonly need to be set to 9600 after following instructions in the ZB to SE Conversion Kit.



**In-Premise Display / Meter Simulator**

XBee Settings | Network | In-Premise Display | Meter | Debug

Connection Status

XBee Status: 0x23 - Valid Coordinator or Routers found, but they are not allowing joining (NJ expired) Refresh

Configure COM Port

Serial Port List

COM1  
COM27  
COM72

Baud: 115200

Open COM Port  
Refresh Port List  
Close COM Port

COM27 - 115200

XBee Settings

XBee Software Version: 0x3319 Hardware Version: 0x1941

Network Protocol: Smart Energy XBee Chipset: Ember

Device Type: router Network Address: [FFFE]

Operating 16-bit PAN ID: 0xFFFF Operating Channel: 0

EUI: [00:13:A2:00:40:15:B0]

Join and Security Parameters

Extended PAN ID: 0x0000000000000000 Set PAN ID

Link Key Set Link Key

Install Code Set Install Code

Attempt to Join Network Enable Joining

Configure Certificate

CA Public Key

Implicit Cert

Private Key

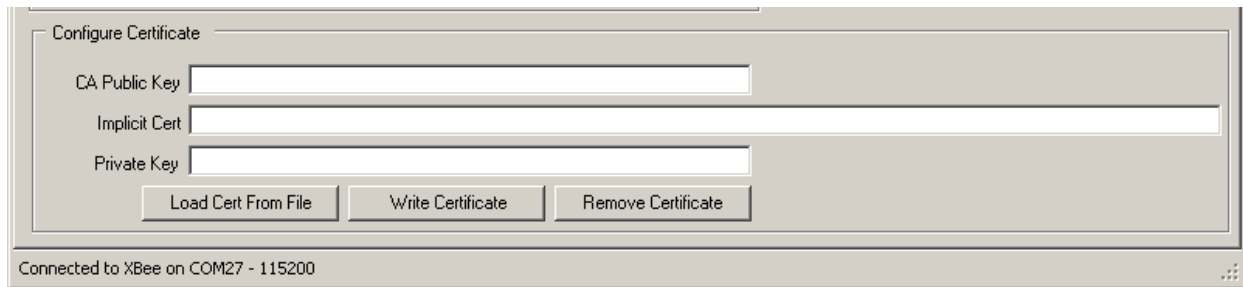
Load Cert From File Write Certificate Remove Certificate

Connected to XBee on COM27 - 115200

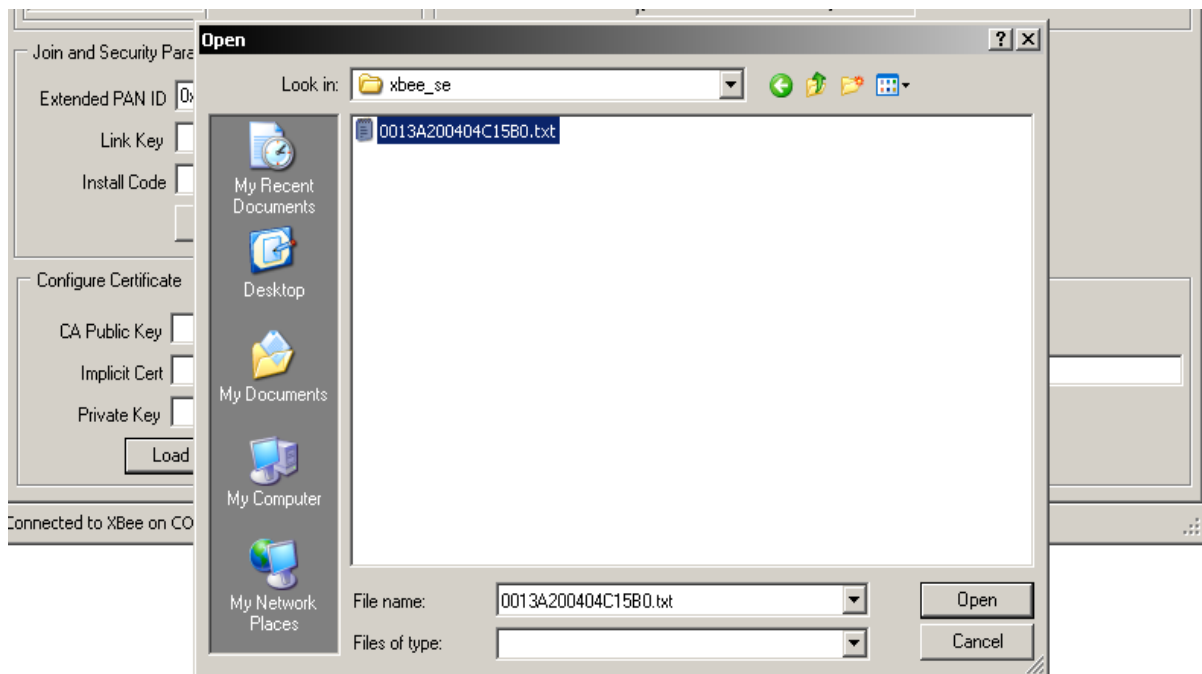
6. Load the test certificate information that corresponds to the serially-attached XBee. This can be accomplished in two ways:

### Load Cert From File

- a. Click **Load Cert From File**.



- b. Select the test certificate file that corresponds to the EUI field under XBee Settings and click **Open**.







## Manual Entry

- a. Manually enter the CA Public Key, Implicit Cert and Private Key.

Configure Certificate

CA Public Key 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8

Implicit Cert 0203a28e336ffb26143e475a6e780809a84a4e487b100013a200404c15c0544553545345434101090010000000000000

Private Key 03f71f529dd5e700a698517668b7a3a97ad59d03f6

Load Cert From File Write Certificate Remove Certificate

Connected to XBee on COM27 - 115200

7. Write the Certificate to the serially-attached XBee by clicking **Write Certificate**.
8. Enter a 128-bit hex value of your choosing into the Link Key field and click **Set Link Key** to write the link key to the serially-attached XBee. Alternately, you may enter Install Code, a 48-, 64-, 96-, or 128-bit hex value (including 16-bit CRC), and click **Set Install Code** to write a link key computed from the install code to the serially-attached XBee.

In-Premise Display / Meter Simulator

XBee Settings Network In-Premise Display Meter Debug

Connection Status

XBee Status 0x23 - Valid Coordinator or Routers found, but they are not allowing joining (NJ expired) Refresh

Configure COM Port

Serial Port List

COM1

COM27

COM72

Baud 115200

Open COM Port

Refresh Port List

Close COM Port

COM27 - 115200

XBee Settings

XBee Software Version 0x3319 Hardware Version 0x1941

Network Protocol Smart Energy XBee Chipset Ember

Device Type router Network Address [FFFF]

Operating 16-bit PAN ID 0xFFFF Operating Channel 0

EUI [00:13:A2:00:40:15:B0]

Join and Security Parameters

Extended PAN ID 0x0000000000000000 Set PAN ID

Link Key ABCDEF1234567890 Set Link Key

Install Code Set Install Code

Attempt to Join Network Enable Joining

Configure Certificate

CA Public Key 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8

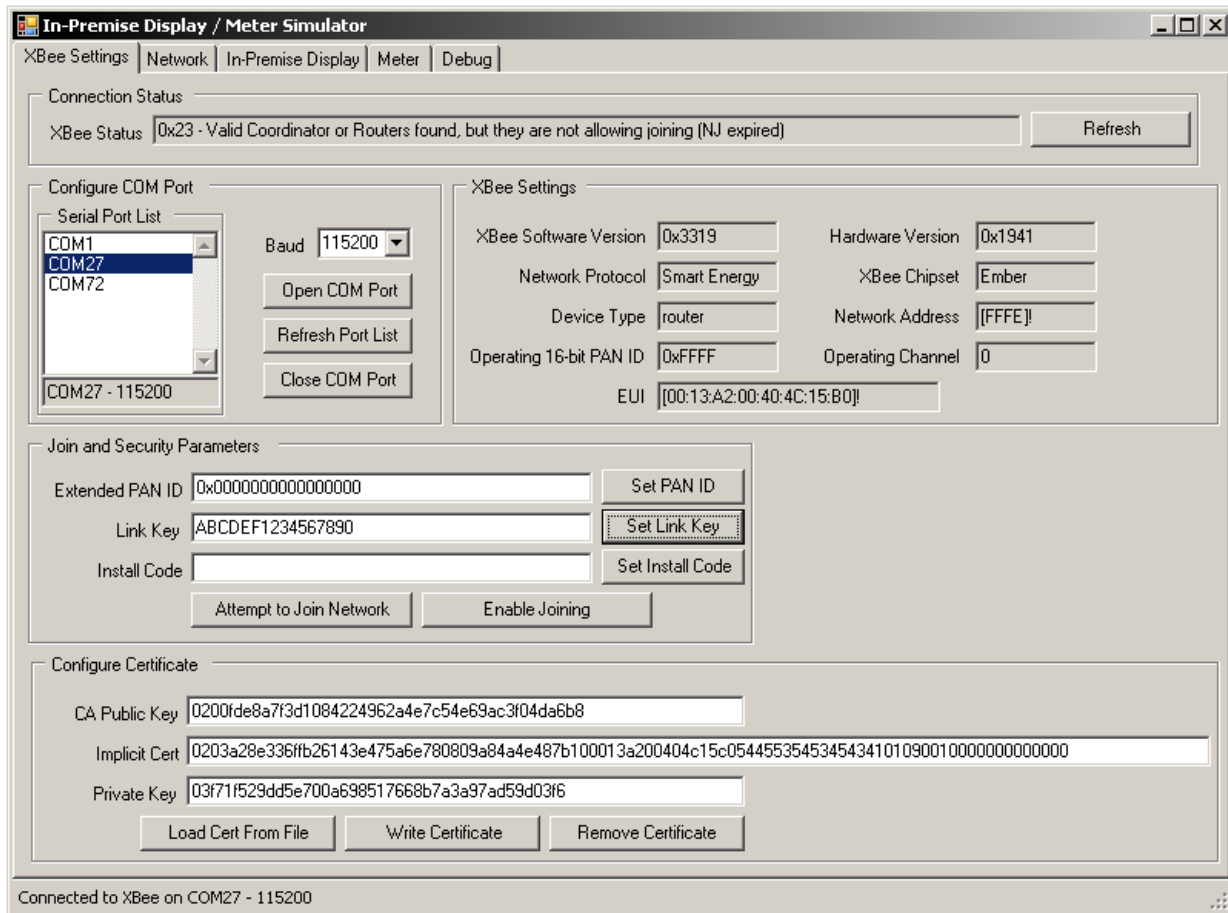
Implicit Cert 0203a28e336ffb26143e475a6e780809a84a4e487b100013a200404c15c0544553545345434101090010000000000000

Private Key 03f71f529dd5e700a698517668b7a3a97ad59d03f6

Load Cert From File Write Certificate Remove Certificate

Connected to XBee on COM27 - 115200

9. Enable joining on the trust center and register the link key / installation code of the XBee. If using an ESI coordinator, send an add\_device RPC request to the gateway in order to allow the serially-attached XBee to join the gateway's network. Set the parameters of the request to correspond to the serially-attached XBee and to enable joining.
10. Click **Attempt to Join Network** and wait until XBee Status indicates that the XBee has joined with the network. The XBee Status field will display status indicators as it attempts to join. In case of failure these indicators may aid in troubleshooting.



**In-Premise Display / Meter Simulator**

XBee Settings | Network | In-Premise Display | Meter | Debug

Connection Status

XBee Status: 0x23 - Valid Coordinator or Routers found, but they are not allowing joining (NJ expired) Refresh

Configure COM Port

Serial Port List: COM1, COM27, COM72

Baud: 115200

Open COM Port Refresh Port List Close COM Port

COM27 - 115200

XBee Settings

XBee Software Version: 0x3319 Hardware Version: 0x1941

Network Protocol: Smart Energy XBee Chipset: Ember

Device Type: router Network Address: [FFFE]

Operating 16-bit PAN ID: 0xFFFF Operating Channel: 0

EUI: [00:13:A2:00:40:4C:15:B0]

Join and Security Parameters

Extended PAN ID: 0x0000000000000000 Set PAN ID

Link Key: ABCDEF1234567890 Set Link Key

Install Code: Set Install Code

Attempt to Join Network Enable Joining

Configure Certificate

CA Public Key: 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8

Implicit Cert: 0203a28e336ffb26143e475a6e780809a84a4e487b100013a200404c15c0544553545345434101090010000000000000

Private Key: 03f71f529dd5e700a698517668b7a3a97ad59d03f6

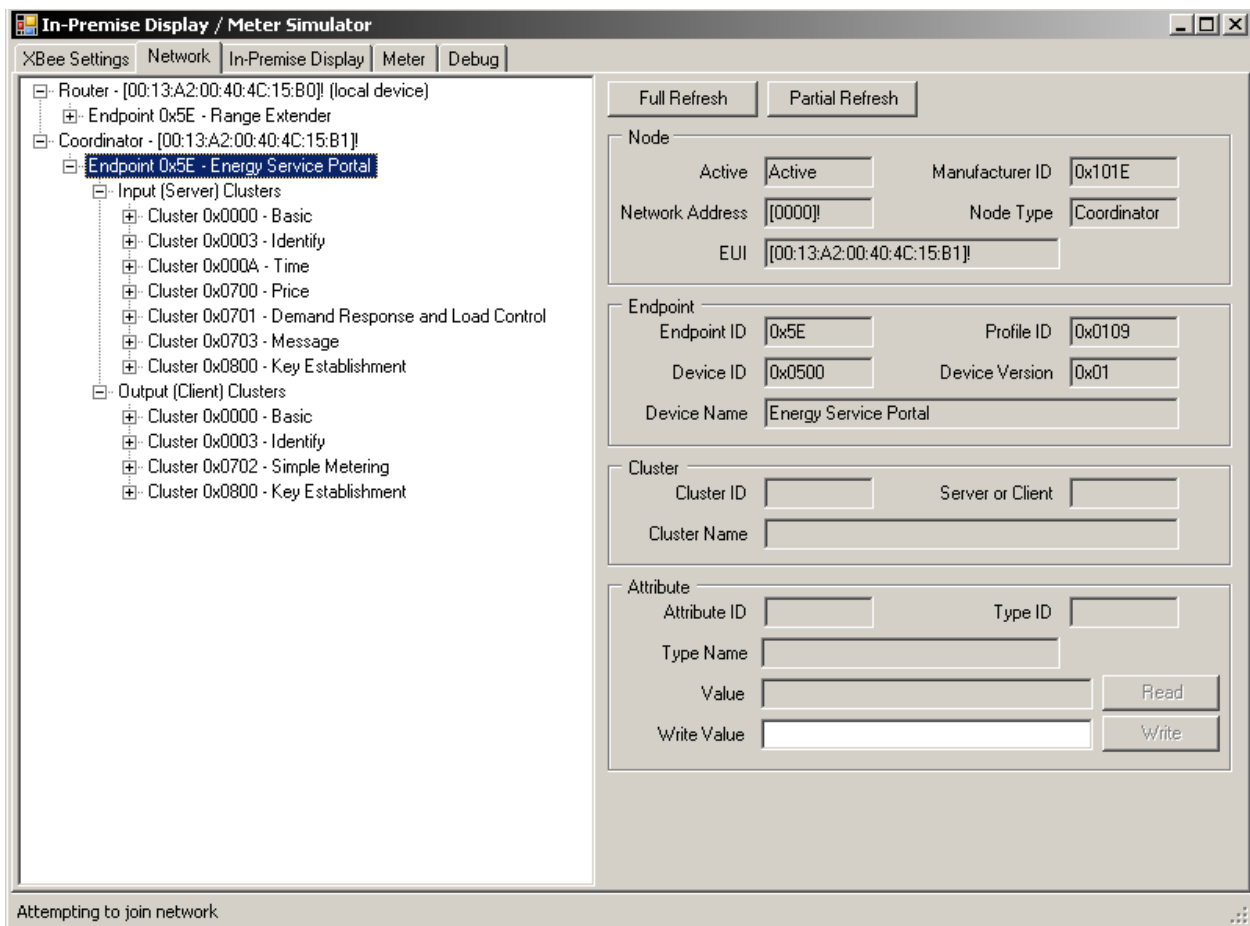
Load Cert From File Write Certificate Remove Certificate

Connected to XBee on COM27 - 115200

## Network View

The Network tab provides a convenient overview of all endpoints and clusters of devices which have been detected on the network. The information displayed should closely match the information returned by a `get_device_information` RPC request sent to the gateway (see “`get_device_information`” on page 74).

To detect new devices, endpoints and clusters click **Partial Refresh**. To clear and rediscover all known device information, click **Full Refresh**.



**In-Premise Display / Meter Simulator**

XBee Settings | **Network** | In-Premise Display | Meter | Debug

Router - [00:13:A2:00:40:15:B0] (local device)

- Endpoint 0x5E - Range Extender
- Coordinator - [00:13:A2:00:40:15:B1]
  - Endpoint 0x5E - Energy Service Portal
    - Input (Server) Clusters
      - Cluster 0x0000 - Basic
      - Cluster 0x0003 - Identify
      - Cluster 0x000A - Time
      - Cluster 0x0700 - Price
      - Cluster 0x0701 - Demand Response and Load Control
      - Cluster 0x0703 - Message
      - Cluster 0x0800 - Key Establishment
    - Output (Client) Clusters
      - Cluster 0x0000 - Basic
      - Cluster 0x0003 - Identify
      - Cluster 0x0702 - Simple Metering
      - Cluster 0x0800 - Key Establishment

**Full Refresh** **Partial Refresh**

**Node**

Active:  Manufacturer ID:   
 Network Address:  Node Type:   
 EUI:

**Endpoint**

Endpoint ID:  Profile ID:   
 Device ID:  Device Version:   
 Device Name:

**Cluster**

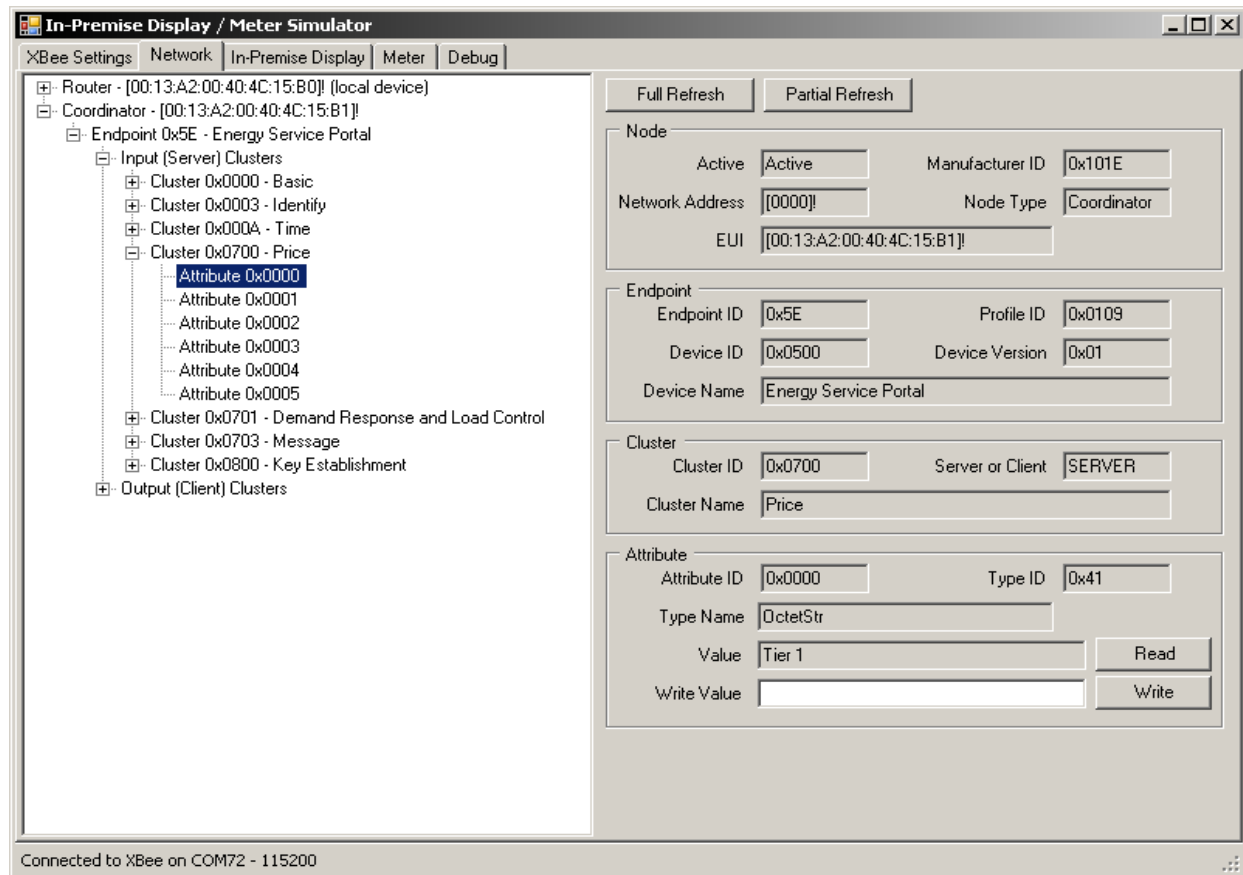
Cluster ID:  Server or Client:   
 Cluster Name:

**Attribute**

Attribute ID:  Type ID:   
 Type Name:   
 Value:    
 Write Value:

Attempting to join network

ZCL attributes can be read and written from the Network Tab after selecting an attribute.



## In-Premise Display

To create an In-Premise Display endpoint, go to the In-Premise Display tab and click the **Enable In-Premise Display** check box. This will create Price, Messaging and Simple Metering client clusters on a new endpoint on the simulated device.

The sample will display the price from any price servers on the network. If there is an ESI gateway on the network, send it a `create_price_event` RPC request to set the current price. Here is an example:

```
<create_price_event>
  <record type="PublishPriceRecord">
    <provider_id>1234</provider_id>
    <issuer_event_id>0xABCD</issuer_event_id>
    <duration_in_minutes>10</duration_in_minutes>
    <price>100</price>
  </record>
```

```
</create_price_event>
```

The price event will automatically be sent to all known Price client clusters. You should see the new price value appear. If there is an Aux Gateway on the network, it should receive the price event and return a “received\_price\_event” (see “received\_price\_event” on page 114) response and an “updated\_active\_price\_event” (see “updated\_active\_price\_event” on page 114), assuming there was not another price event with a matching issuer\_event\_id.



The sample will also display any active messages on the network. If there is an ESI gateway on the network, send it a create\_message\_event RPC request to display a message. Here is an example:

```
<create_message_event synchronous="true">
  <record type="DisplayMessageRecord">
    <message_id>1234</message_id>
    <duration_in_minutes>5</duration_in_minutes>
    <message type="string">NOBODY expects the Spanish Inquisition!</message>
  </record>
```

```
</create_message_event>
```

The message event will automatically be sent to all known Messaging client clusters. You should see the new message appear. If there is an Aux Gateway on the network, it should receive the message event and return a “received\_message\_event” (see “received\_message\_event” on page 104) response and an “updated\_active\_message\_event” (see “updated\_active\_message\_event” on page 105).

Note: The message server cluster will automatically discard messages with the same message\_id, so be sure to use unique message\_ids when creating message events.





Additionally, if a Simple Metering server cluster is found on the network, the current usage information will be automatically retrieved and displayed via ZCL reporting.

The screenshot shows the 'In-Premise Display / Meter Simulator' application window. It features a tabbed interface with 'XBee Settings', 'Network', 'In-Premise Display' (selected), 'Meter', and 'Debug'. The 'In-Premise Display' tab contains several sections:

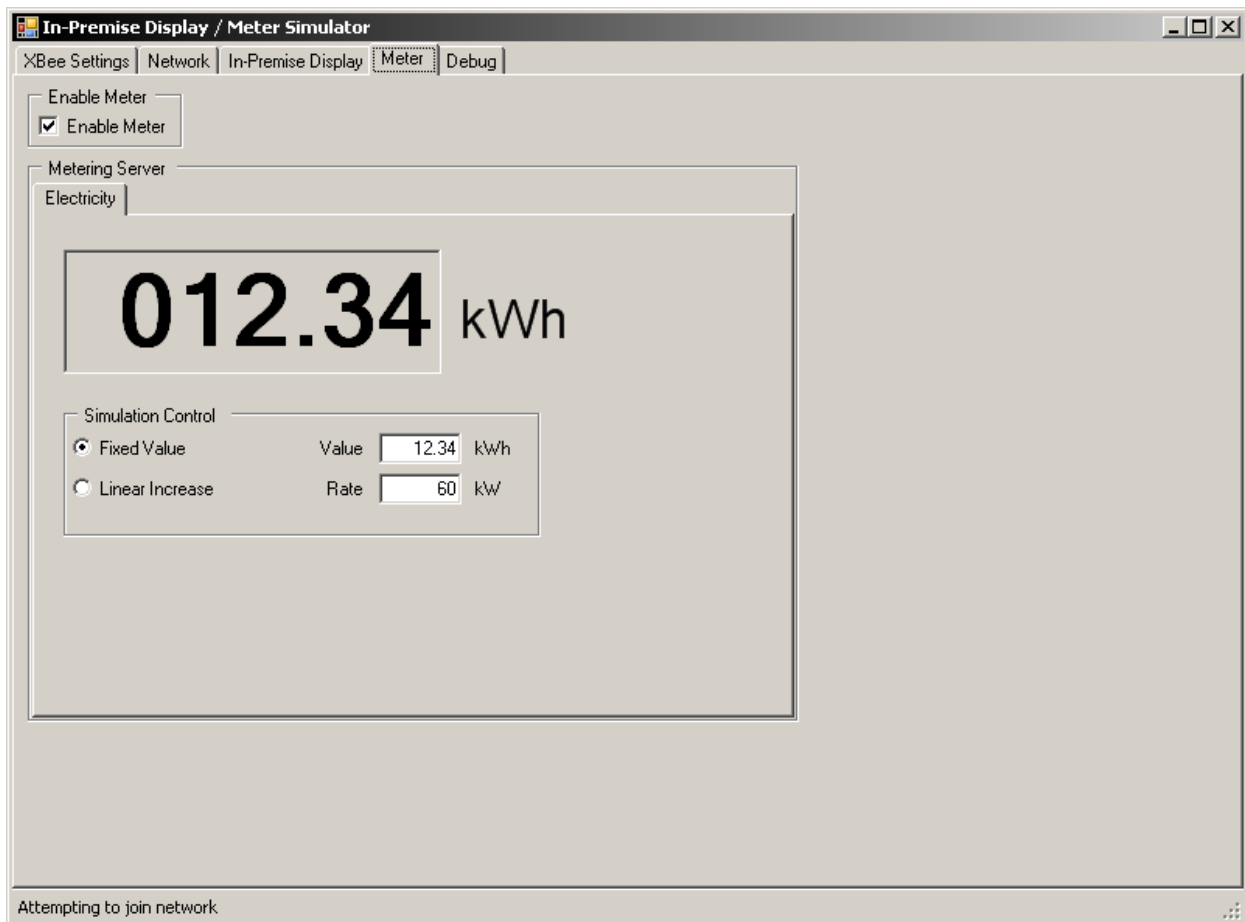
- Enable Display:** A checkbox labeled 'Enable In-Premise Display' is checked.
- Time:** Two text boxes show 'PC Time' as 'Mon Nov 02 15:55:43 2009' and 'UTC Time' as '310517743'.
- Price Client:** A sub-tab labeled 'Electricity' displays a large digital readout showing '\$1.00 kWh'. Below this, there are four input fields: 'Event ID' (0xABCD), 'Rate Label' (empty), 'Start Time' (310517662), and 'End Time' (310518262).
- Metering Client:** A sub-tab labeled 'Electricity' displays a large digital readout showing '012.34 kWh'.
- Message Client:** A text box at the bottom contains the message 'NOBODY expects the Spanish Inquisition!' with a 'Confirm' button to its right.

A status bar at the bottom of the window indicates 'Attempting to join network'.

## Meter

To create a Metering Device endpoint, go to the Meter tab and click the **Enable Meter** check box. This will create a Simple Metering server cluster on a new endpoint on the simulated device.

You may set the meter to either have a fixed usage value or a linearly increasing value.







## *CHAPTER 2*

# *General Operation*

The following section describes the standard initialization, automation, and configuration features of the gateway.



## STARTUP SEQUENCE

The following operations are performed in order by the gateway when powered on.

1. Core modules are imported and core objects are initialized. The gateway will then load the modules and objects specified in the configuration files (see “Global Saved Files” on page 48). All RPC requests are now registered and available for use. Be aware that some RPC requests require time to be synchronized and / or the gateway to be joined to a network and may return errors.
2. If enabled, the gateway synchronizes time with an NTP server (see “Registry” on page 45). By default, the ESI coordinator and ESI router will enable NTP; however, the Aux Gateway will try to set its time using the ZCL time client cluster after it has joined the network.
3. The gateway waits for its XBee radio to report that it is joined to a network. Until the radio reports that it is joined to a network, all outgoing ZigBee messages will return an error.
4. The gateway is fully operational.



## REMOTE DEVICE MANAGEMENT

### Device Detection

A remote device is detected when one of the following happens:

- A ZigBee message is received by the gateway from that device.
- A ZigBee message is successfully sent to that device from the gateway.

If the remote device is in the list of registered devices, or if the gateway is in open joining mode (see “Explicit Device Add and Open Join” on page 43), ZDO queries will be sent to the remote device in the following order:

- A Match Descriptor request for the SE key establishment cluster server (0x0800) and Basic cluster (0x0000). This will update the known network address of the remote device. A device which responds to at least this request will be considered active.
- A Node Descriptor request.
- A Power Descriptor request.
- An Active Endpoints request.

For each endpoint returned by the Active Endpoints request, a Simple Descriptor request will also be sent to find all server and client clusters on the endpoint. Any server clusters on the gateway will be notified of matching client clusters on the remote device, and likewise, any client clusters on the gateway will be notified of matching server clusters on the remote device.

### SE Client Clusters and Active Devices

If an SE client cluster on the gateway is notified of a server cluster, that server cluster will be added to the list of known SE server clusters for that particular SE client cluster. The client cluster uses this list to request the currently active events from the server(s).

### SE Server Clusters and Active Devices

If an SE server cluster on the gateway is notified of a client cluster, that client cluster will be added to the list of known SE client clusters for that particular SE server cluster. The server cluster uses this list to unicast certain SE commands to all clients.

SE events stored by the gateway’s server cluster not yet received by a device will be sent automatically when a device becomes active. If the client also requests events from the server, these events may be sent to the client twice. However, the gateway will not consider this case to be an error.



## Transmission Retries and Device Inactivity

A remote device is considered inactive if a certain number of sequential transmissions to that device fail (see “ZDO\_Device\_Manager.max\_sequential\_TX\_failures” on page 46). When a remote device is marked as inactive, clusters on the gateway will be informed that any matching clusters on the remote device are no longer available.

## SE Server Clusters and Inactive Devices

If an SE server cluster on the gateway is notified that a client cluster has been removed, that client cluster will be removed from the list of known SE client clusters for that particular SE server cluster. Events which are missed by the remote device while it is inactive will be sent when that device becomes active again.

## Periodic Refresh

In order to keep network information up to date, refresh messages are sent periodically (see “ZDO\_Device\_Manager.refresh\_rate” on page 46). These are in the form of Match Descriptor broadcast requests for the SE Key Establishment cluster server (0x0800). Devices which do not respond within the timeout period will be considered to have a transmission failure (see “Conversation.TX\_status\_timeout” on page 45).

In this manner, active devices which have dropped off the network or otherwise become unable to receive transmissions (interference, power loss, etc), will be marked as inactive even if no user-initiated communication to that device has taken place. Devices currently marked as inactive will be marked as active if they respond to the broadcast.

## Explicit Device Add and Open Join

The gateway can be in either open join or explicit device add mode. When in open join mode, any device can become active. Devices that become active will be added to the list of known devices (see “Device Detection” on page 42). When in explicit device add mode, only devices which have previously been added as known devices can become active. In either mode, devices can be added to the list of known devices by using the add\_device command (see “add\_device\_response Parameters:” on page 73).

The gateway defaults to open join mode.  
(See “ZDO\_Device\_Manager.require\_explicit\_device\_add” on page 47.)

## ZCL Reporting and Device Activity

If ZCL attribute reporting is configured on the gateway to expect reports from an active remote device, and that device has not sent a report within the configuration's timeout setting, the reporting configuration will be resent to the remote device. The remote device may have had a power loss or other fatal error and so lost its reporting configuration.

ZCL attribute reports will not be sent to a device which is inactive.

The gateway also supports a simulated reporting mode called "pseudo reporting". Pseudo reporting acts like normal ZCL reporting, but instead of sending a Reporting Configuration to the remote device, the gateway will send periodic reads instead. The gateway keeps track of previous read values and will send an `attribute_report` RPC response (see "attribute\_report" on page 83) using the min/max reporting interval and delta change, as if the remote device had been configured with reporting. This allows you to use the reporting interface on devices that do not support reporting. The gateway sends a read to the remote device every Min Reporting Interval. Pseudo reporting is enabled by setting the `pseudo_reporting` tag to `TRUE` in the `start_receiving_reports` RPC request (see "start\_receiving\_reports" on page 82).

## REGISTRY

The registry is created when the gateway first boots and contains power-safe global settings to control gateway behavior. Individual entries can be accessed and modified through the `registry_configuration` RPC request. A help RPC request can be used to retrieve the entire registry. If necessary, manually editing `registry_settings.ini` and rebooting is possible but not recommended.

### Settings and Defaults

Registry Setting	Default	Description
<b>Conversation.default_timeout</b>	40	For unicast conversations with an expected response message, if that response has not arrived within this many seconds a time-out has occurred. For broadcast conversations, the conversation will automatically terminate after this many seconds.
<b>Conversation.TX_status_timeout</b>	10	If the XBee does not provide a transmission status for an out-bound message within this many seconds, the transmission is assumed to have failed.
<b>Endpoint.max_socket_retries</b>	3	If an exception occurs while sending to the socket, the send will be retried up to this many times.
<b>NTP_client.server1</b>	north-amer-ica.pool.ntp.org	The main NTP server that will be used for time synchronization.
<b>NTP_client.server2</b>	pool.ntp.org	A backup NTP server that will be used if synchronization with the first server fails.
<b>NTP_client.sync_rate</b>	86400	How often the gateway's time is synchronized with an NTP server, in seconds.
<b>RPC_General_Interface.debug_print_severity</b>	0	If a message is generated by the firmware with severity greater than or equal to this value, it will be printed to the screen. Default to all messages.

Registry Setting	Default	Description
<b>RPC_General_Interface.debug_rpc_severity</b>	1	If a message is generated by the firmware with severity greater than or equal to this value, it will be sent as an RPC response. Default to all warnings and errors.
<b>RPC_Manager.max_buffer_items</b>	250	Up to this many outgoing RPC responses will be stored by the gateway before the oldest response is discarded.
<b>RPC_Manager.pushed_file_extension</b>	xml	If pushing is enabled, this string will be used as the file extension of the pushed file.
<b>RPC_Manager.pushed_file_prefix</b>	RPC_Response	If RPC_Manager.use_push is TRUE, this string will be used as the first part of filename for the pushed file.
<b>RPC_Manager.use_push</b>	FALSE	If TRUE, RPC responses will be automatically pushed to iDigi. (See “Automatic Response Pushing” on page 54.)
<b>RPC_Manager.use_timestamps</b>	TRUE	If TRUE, all RPC responses will include a timestamp indicating when they were generated. Timestamps are in standard Unix time (seconds since 1970).
<b>ZCL_Cluster.disable_default_response</b>	TRUE	If TRUE, the disable default response bit of ZCL messages generated by the gateway will be set to 1. If FALSE the bit will be set to 0.
<b>ZDO_Device_Manager.max_sequential_TX_failures</b>	3	If this many transmission failures to a particular device occur in a row, that device will be marked as inactive. (See “Transmission Retries and Device Inactivity” on page 43.)
<b>ZDO_Device_Manager.refresh_rate</b>	120	Every this many seconds a match descriptor broadcast is sent. (See “Periodic Refresh” on page 43.)

Registry Setting	Default	Description
<b>ZDO_Device_Manager.require_explicit_device_add</b>	FALSE	If TRUE, devices must be explicitly added before they can become active. If FALSE, devices can become active without being explicitly added. If not specified, the gateway will use its default behavior. (See “Explicit Device Add and Open Join” on page 43.)
<b>Cluster.disable_APS_encryption</b>	FALSE	If TRUE, APS encryption checks for clusters which normally require encryption will be ignored. For normal operation, this should be set to FALSE. Note that changing this value may result in the XBee leaving its current network.



## POWER SAFETY

Certain information is automatically saved to flash in order to maintain gateway functionality in case of a power failure.

### Global Saved Files

The gateway stores the following files under /WEB/python/ for persistent configuration and startup behavior. These files can be viewed through iDigi, the web interface, or built in file system commands of the gateway.

File	Description
<b>registry_settings.ini</b>	<p>Stores the current state of the registry settings for the gateway. (See “registry_configuration” on page 63.) Each line of the registry file is a single entry and is formatted as follows where type is int, bool, or str.</p> <p>Format:      registry_name [type registry_value]</p> <p>If a type and registry_value are not given then the registry entry exists but has no value.</p> <p><b>Sample:      Cluster.disable_APS_encryption bool False</b></p>
<b>devices.ini</b>	<p>Stores the 64-bit extended address of all known devices. Any device added via the add_device RPC request will be added to this file. If the gateway is in open joining mode, any new devices which become active will also be added to this file. Devices will only be removed from this file via the remove_device RPC request.</p> <p>Note:      It is possible to modify this file directly. However, when the gateway is operating as a trust center, its wireless radio also maintains an internal table of link keys. Adding a device to devices.ini directly does not update the wireless radio's internal table and the new device may not be able to join. The add_device and remove_device RPC requests update the internal table and are the preferred interface to devices.ini.</p>
<b>modules.ini</b>	<p>Specifies additional modules to be imported on startup. (See “add_module” on page 64 and “remove_module” on page 64.)</p> <p><b>Sample:      RPC_ZCL_Interface RPC_SE_Interface</b></p> <p>This sample will import the modules RPC_ZCL_Interface.pyc and RPC_SE_Interface.pyc on startup.</p>
<b>interfaces.ini</b>	<p>Specifies additional interface classes to be instantiated on startup. (See “add_interface” on page 65 and “remove_interface” on page 65.)</p> <p><b>Sample:      RPC_ZCL_Interface RPC_SE_Interface</b></p> <p>This will instantiate the classes RPC_ZCL_Interface and RPC_SE_Interface on startup.</p>

File	Description
<b>endpoints.ini</b>	<p>Specifies additional endpoints to be instantiated on startup. Optional parameters default to the class definition. (See “add_endpoint” on page 66 and “remove_endpoint” on page 67.)</p> <p>Format:      endpoint_class_name endpoint_id [profile_id [device_id [device_version]]]</p> <p><b>Sample:      SE_EnergyServicePortal 0x5E</b></p> <p>This will instantiate an Energy Service Portal on endpoint 0x5E on startup.</p>
<b>clusters.ini</b>	<p>Specifies additional clusters to be instantiated on startup for a given endpoint. Optional parameters default to the class definition. (See “add_cluster” on page 67 and “remove_cluster” on page 68.)</p> <p>Format:      cluster_class_name endpoint_id [server_or_client [cluster_id [enable_APS_encryption]]]</p> <p><b>Sample:      ZCL_Cluster 0x5E 1 0x0200</b></p> <p>This will instantiate a generic ZCL Cluster object on endpoint 0x5E as a server with cluster ID 0x0200.</p>

## Hidden Saved Files

Aliases and certain cluster specific information are stored in subdirectories under /WEB/python/. These files are not accessible through the web interface, not intended to be human-readable, and are frequently created and removed as needed during normal operation.

Type	Description
<b>RPC aliases</b>	Any RPC aliases which have been added will be saved. (See “Aliases” on page 60.)
<b>SE Events (SE server clusters only)</b>	Valid SE events are saved on the server. Only the event itself is saved, incidental information such as what devices have received a particular event are not saved. When events terminate their files are removed.
<b>ZCL Reporting Configurations</b>	When the gateway is expecting reports it will save the reporting configuration to a file. The file will be removed when reports are stopped.
<b>Power-safe ZCL Attributes</b>	When ZCL attributes are created they can be declared as power-safe. Power-safe attributes are saved to file when written to a non default value. All SE cluster attributes are declared power-safe.

---

## *CHAPTER 3      Communication Protocol*

The following section explains the basics of communicating with the gateway when it is running the SE framework. An API reference for all RPC requests and responses can be found in “Appendix A” on page 63.

### **BASIC COMMUNICATION OVERVIEW**

The gateway communicates via XML encoded RPC requests and responses. These messages are sent within an RCI wrapper that specifies the RPC target. Typically requests and responses will also be routed through iDigi which specifies a further SCI wrapper. Each request can specify whether it should block until a response is ready or asynchronously return the response. By default, the gateway operates in a polled mode where asynchronous responses are cached locally and must be explicitly requested. The gateway can also be configured to push asynchronous responses to the iDigi servers as they are generated.

In order to communicate with the gateway, XML requests must be generated along with RCI/SCI wrappers to address the correct device. It is outside the scope of this document to cover methods for generating and parsing XML, sending requests to the gateway, or communicating with the iDigi server from within a custom application. The “iDigi SE Web Sample, Communicating with Gateway” on page 24 can be used to communicate with the gateway through iDigi.

## RPC Request and Response Example

RPC requests and responses are contained in an RCI wrapper, which specifies the command target (i.e. "RPC\_request"). This RCI wrapper is contained in an SCI wrapper, which specifies devices to which the commands will be sent. Here is an example to send a synchronous get\_time request and an asynchronous get\_version request to two gateways:

```
<sci_request version="1.0">
  <send_message>
    <targets>
      <device id="00000000-00000000-001122FF-FF334455"/>
      <device id="00000000-00000000-001122FF-FF334456"/>
    </targets>
    <rci_request version="1.1">
      <do_command target="RPC_request">
        <get_version />
        <get_time synchronous="TRUE" />
      </do_command>
    </rci_request>
  </send_message>
</sci_request>
```



The responses from multiple devices will be accumulated into a single SCI response by iDigi. The reply to the above example would look like the following:

```
<sci_reply version="1.0">
  <send_message>
    <device id="00000000-00000000-001122FF-FF334455"/>
      <rci_reply version="1.1">
        <do_command>
          <get_time_response timestamp="1234567888.0"/>
            <UTC_2000>0x133FDDDD<UTC_2000><UTC_1970>0x4BAD215D<UTC_1970>
          </get_time_response>
        </do_command>
      </rci_reply></device>
    <device id="00000000-00000000-001122FF-FF334456"/>
      <rci_reply version="1.1">
        <do_command>
          <get_time_response timestamp="1234567888.0"/>
            <UTC_2000>0x133FDDDD<UTC_2000><UTC_1970>0x4BAD215D<UTC_1970>
          </get_time_response>
        </do_command>
      </rci_reply></device>
```

To retrieve asynchronous response messages from the gateway, send to the "RPC\_response" target. Leaving off the SCI wrapper, the RCI wrapper and command looks like the following:

```
<rci_request version="1.1">
  <do_command target="RPC_response">
    <get_responses />
  </do_command>
</rci_request>
```

A limit can be set on the number of responses returned by specifying the "limit" attribute. Responses will be returned starting with the oldest.

```
<rci_request version="1.1">
  <do_command target="RPC_response">
    <get_responses limit="10"/>
  </do_command>
</rci_request>
```



The reply to an RPC\_response request contains a listing of RPC responses in chronological order. By default timestamps are enabled and every response will indicate when it was generated. The top level “responses” tag will also indicate how many response messages remain in the gateway’s buffer. For example:

```
<rci_reply version="1.1">
  <do_command>
    <responses remaining="0" timestamp="1234567890.0">
      <message timestamp="1234567888.0">
        <description type="string">You must bring us another shrubbery.</description>
        <severity>0x1</severity>
      </message>
      <get_version_response timestamp="1234567889.0">
        <version type="string">1.3.0</version>
        <version_extended type="dict">
          <SE_spec_version type="string">1.0 revision 15</SE_spec_version>
          <description type="string">SE Framework with support for ESI and IPD
            functionality.</description>
        </version_extended>
      </get_version_response>
    </responses>
  </do_command>
</rci_reply>
```



## Automatic Response Pushing

The gateway can be configured to push responses immediately to the iDigi server instead of the default polling mode via `RPC_response`. (See “`RPC_Manager.use_push`” on page 46.) Memory on the gateway is limited and it can be advantageous to store responses on the iDigi server to avoid the risk of dropping messages if too many responses are generated.

**Note:** Response messages will not be removed from the gateway’s buffer until they have been successfully pushed to the iDigi server. Responses can still be retrieved via `RPC_response` if pushing does not succeed for any reason.

## File Format

Each pushed file contains a single response without any RCI or SCI wrapper.

```
<message>
  <description type="string">
    You must bring us another shrubbery.
  </description>
  <severity>0x1</severity>
</message>
```

The filename is formatted with the following information; pushed file prefix (defaults to `RPC_response`), timestamp, sequence number, response type, and file extension (defaults to `xml`). The sequence number differentiates responses pushed in the same second.

```
RPC_response-1234567890-0000-message.xml
RPC_response-1234567890-0001-get_version_response.xml
RPC_response-1234567890-0002-get_device_information_response.xml
RPC_response-1234567891-0000-message.xml
```

## XML RPC INTERFACE OVERVIEW

### Conversion to Method Call

XML RPC requests convert into method calls on the gateway in a straightforward manner.

#### Example

**An XML RPC request like this:**

```
<request_name>
  <param1>value1</param1>
  <param2>value2</param2>
  <param3>value3</param3>
</request_name>
```

**Will result in an method call like this:**

```
request_name(param1 = value1, param2 = value2, param3 = value3)
```

XML RPC responses similarly specify the name of the response and have a collection of parameters.

#### Example

**An XML RPC response might look like this:**

```
<response_name>
  <param1>value1</param1>
  <param2>value2</param2>
  <param3>value3</param3>
</response_name>
```



## Parameter Type Specification Overview

The type of a parameter is specified by the attribute “type” within the tag.

### Example

**param1 is specified as being of type string:**

```
<param1 type="string">My favorite color is blue.</param1>
```

If no type is specified, the value will be converted to a **bool**, **int** or **float**, in that order.

### Example

<code>&lt;param1&gt;TRUE&lt;/param1&gt;</code>	param1 is a <b>bool</b>
<code>&lt;param1&gt;1&lt;/param1&gt;</code>	param1 is an <b>int</b>
<code>&lt;param1&gt;1.0&lt;/param1&gt;</code>	param1 is a <b>float</b>

For responses, type will always be specified.

## Simple Parameter Types

<b>int</b>	<b>example:</b> <code>&lt;param type="int"&gt;42&lt;/param&gt;</code> <code>&lt;param type="int"&gt;0xFF&lt;/param&gt;</code>
<b>float</b>	<b>example:</b> <code>&lt;param type="float"&gt;3.14159&lt;/param&gt;</code>
<b>bool</b>	Takes "TRUE" or "FALSE", case insensitive.  <b>example:</b> <code>&lt;param type="bool"&gt;TRUE&lt;/param&gt;</code>
<b>string</b>	A character string.  <b>example:</b> <code>&lt;param type="string"&gt;Tis but a scratch.&lt;/param&gt;</code>
<b>MAC</b>	Intended for use with long address strings, which commonly contain delimiter characters.  <b>example:</b> <code>&lt;param type="MAC"&gt;00:11:22:33:44:55:66:77&lt;/param&gt;</code>
<b>base16</b>	A base 16-encoded binary string.  <b>example:</b> <code>&lt;param type="base16"&gt;1234567890ABCDEF&lt;/param&gt;</code>
<b>base64</b>	A base 64-encoded binary string.  <b>example:</b> <code>&lt;param type="base64"&gt;3q2+78r+&lt;/param&gt;</code>
<b>none</b>	Used to provide a parameter but with no type or value. Some requests may differentiate between a missing parameter and a none parameter. Unlike other parameters, do not include the "type" attribute or value.  <b>example:</b> <code>&lt;param/&gt;</code>

## Complex Parameter Types

<b>list</b>	<p>A list of subparameters. Note that because a list is an anonymous data structure, the tag names of subparameters are ignored.</p> <p><b>example:</b></p> <pre>&lt;param type="list"&gt;   &lt;item&gt;42&lt;/item&gt;   &lt;item&gt;FALSE&lt;/item&gt;   &lt;item type="string"&gt;Camelot!&lt;/item&gt; &lt;/param&gt;</pre> <p><b>would generate:</b></p> <pre>param=[42, False, "Camelot!"]</pre>
<b>dict</b>	<p>A dictionary of subparameters. The tag names of the subparameters will become the keys of the dictionary.</p> <p><b>example:</b></p> <pre>&lt;param type="dict"&gt;   &lt;first&gt;1&lt;/first&gt;   &lt;second&gt;TRUE&lt;/second&gt;   &lt;third type="string"&gt;It's only a model.&lt;/third&gt; &lt;/param&gt;</pre> <p><b>would generate:</b></p> <pre>param={"first" : 1, "second" : True, "third" : "It's only a   model."}</pre>

<b>record</b>	<p>A record object with specific subparameters. The subparameters will be passed to the record object's constructor according to tag name. (See "Record Reference" on page 116.)</p> <p><b>example:</b></p> <pre>&lt;param type="MagicRecord"&gt;   &lt;foo&gt;9000&lt;/foo&gt;   &lt;bar type="string"&gt;You must bring us...&lt;/bar&gt;   &lt;a_shrubbery type="bool"&gt;TRUE&lt;/a_shrubbery&gt; &lt;/param&gt;</pre> <p><b>would generate:</b></p> <pre>param=MagicRecord(foo = 9000, bar = "You must bring us...",   a_shrubbery = TRUE)</pre>
<b>xml</b>	<p>A way to specify that the enclosed XML is to be preserved as XML and not parsed into types. The XML will be passed into the function as an ElementTree (a Python object that can parse and represent XML).</p> <p><b>example:</b></p> <pre>&lt;param type="xml"&gt;   &lt;dead&gt;     &lt;parrot&gt;This is a late parrot! It's a stiff! Bereft of life,     it rests in peace!&lt;/parrot&gt;   &lt;/dead&gt; &lt;/param&gt;</pre> <p><b>would generate:</b></p> <pre>param=ElementTree.parsestring("""&lt;root&gt;&lt;dead&gt;&lt;parrot&gt;This is a   late parrot! It's a stiff! Bereft of life, it rests in   peace!&lt;/parrot&gt;&lt;/dead&gt;&lt;/root&gt;""")</pre>

## Aliases

XML RPC requests can be abstracted using aliases. To use an alias which has been defined, use the “alias” type. Alias replacements are performed in place.

### Example

**If there was an alias named “EXAMPLE\_ALIAS” defined with the following XML content:**

```
<param1>value</param1>
<param2>value</param2>
<param3>value</param3>
```

**The following XML:**

```
<request_name>
  <EXAMPLE_ALIAS type="alias" / >
</request_name>
```

**Would then become:**

```
<request_name>
  <param1>value</param1>
  <param2>value</param2>
  <param3>value</param3>
</request_name>
```

The main value of aliasing is in generalizing parameters to commands which are sent to a large number of devices.

### Example

**If there are two gateways with the alias “THERMOSTAT” defined:**

**Gateway 1:**

```
<destination_address type="MAC">00:11:22:33:44:55:66:77
</destination_address>
```

**Gateway 2:**

```
<destination_address type="MAC">00:11:22:33:44:AA:BB:CC
</destination_address>
```



**The request that needs to be performed has the following structure:**

```
<request_name>
  <destination_address>value</destination_address>
  <param1>value</param1>
  <param2>value</param2>
</request_name>
```

Ordinarily a different request would be sent to each gateway, since the thermostat addresses on the two networks are different. However, since a THERMOSTAT alias is defined on each gateway for its local thermostat, the same request can be sent to both.

```
<request_name>
  <THERMOSTAT type="alias">
  <param1>value</param1>
  <param2>value</param2>
</request_name>
```

To manage aliases defined on the gateway, use the `add_alias`, `remove_alias`, and `list_aliases` RPC requests. (See “`add_alias`” on page 114, “`remove_alias`” on page 114 and “`list_aliases`” on page 115.)

## Request Identifier

All RPC requests take an optional `request_identifier` parameter which can be used to match requests with later responses. The request identifier can be of any simple type but typically is an integer.

## Broadcasts

RPC requests that specify a destination address can sometimes send the corresponding ZigBee command as a broadcast rather than a unicast. For example, the ZCL Identify command is either unicast to a single device or broadcast to all devices on the network. To specify a broadcast, set the destination address to `00:00:00:00:00:00:FF:FF`.

## Synchronous Requests

Any RPC request sent to the gateway can be made synchronous by adding the attribute `synchronous="TRUE"` to the RPC request tag. Synchronous requests block until the response is generated or a timeout occurs (See the definition entry for `RPC_Manager.synchronous_request_timeout` in the Registry on page 45.) The response is then returned in the RCI Reply instead of being pushed to iDigi or queued in an asynchronous response buffer. In the case of a timeout, the response will be returned asynchronously. Only one request can be processed at a time, and the RCI target



RPC\_request will not be available while a synchronous request is blocking. (For an example using synchronous requests, see RPC Request and Response Example on page 51.)

## *Appendix A*

### XML RPC INTERFACE REFERENCE

#### General Requests/Responses

##### **registry\_configuration**

Sets or gets a registry entry. Can only be used to access an existing entry, not to create a new one.

##### **registry\_configuration Parameters:**

Parameter	Type	Description
name	string	Name of the registry entry to access.
value (optional)	bool, int, float, string or none	If provided, this will set registry name to value. If not provided, this will get the value of registry entry in the response.

##### **registry\_configuration\_response Parameters:**

Parameter	Type	Description
name	string	Name of the registry entry.
write_status (optional)	bool	Only included if the original request was a set request. If TRUE, registry entry successfully updated. If FALSE, registry entry not updated.
write_errors (optional)	string	Only included if the original request was a set request and write_status indicates an error. Description of any errors encountered while setting the registry entry.
value	bool, int, string, or none	Value of registry entry if getting value or original value of request if setting value.



## add\_module

Adds a module without stopping gateway execution. The module is scanned after it has been added and any interface, endpoint, cluster, or record classes are immediately available for use in other interface commands. The module will also be added to modules.ini so that on startup the module will still be imported.

**Note:** The module's .py, .pyc or .pyo file must first be uploaded to the gateway before it can be added by this method. A .py file takes significantly more memory than a .pyc or .pyo file. If your gateway is running low on memory consider precompiling added python modules.

### add\_module Parameters:

Parameter	Type	Description
module_name	string	Name of the module to be imported (do not include file extension).

### add\_module\_response Parameters:

Parameter	Type	Description
module_name	string	Name of the module that was imported.

## remove\_module

Removes the module from modules.ini so that it will not be imported on startup. Modules not imported from modules.ini on startup or via the add\_module command cannot be removed by this request. If the module provided is not currently loaded then no error is indicated but the module will still be removed from modules.ini if present. Note that this will not delete the corresponding .py, .pyc or .pyo file from the gateway. The module will not be removed from RAM unless the Python program is restarted through reboot or the exit RPC command (see exit on page 71).

### remove\_module Parameters:

Parameter	Type	Description
module_name	string	Name of the module to be removed (do not include file extension).

### remove\_module\_response Parameters:

Parameter	Type	Description
module_name	string	Name of the module that was removed.

## add\_interface

Instantiates an interface of the given class, making all public methods inside that class available to be called via RPC. The interface will also be added to interfaces.ini so that on startup the interface will still be available.

### add\_interface Parameters:

Parameter	Type	Description
interface_class	string	Name of the interface class to be instantiated.

### add\_interface\_response Parameters:

Parameter	Type	Description
interface_class	string	Name of the interface class that was instantiated.

## remove\_interface

Removes an interface of the given class, making all public methods inside that class unavailable to be called via RPC. Also removes the interface from interfaces.ini so that it will not be made available at startup.

### remove\_interface Parameters:

Parameter	Type	Description
interface_class	string	Name of the interface class to be removed.

### remove\_interface\_response Parameters:

Parameter	Type	Description
interface_class	string	Name of the interface class that was removed.

## add\_endpoint

Instantiates an endpoint of the given class, along with any of its default clusters. If no endpoint ID is provided, the next available ID will be chosen. Optional parameters will override defaults of the endpoint class. The endpoint will also be added to endpoints.ini so that on startup the endpoint will be instantiated.

### add\_endpoint Parameters:

Parameter	Type	Description
endpoint_class	string	Name of the endpoint class to be instantiated.
endpoint_id (optional)	int	8-bit identifier of the endpoint. By default the next available identifier will be chosen.
profile_id (optional)	int	16-bit profile identifier of the endpoint. Required if the endpoint class does not provide a default.
device_id (optional)	int	16-bit device identifier of the endpoint.
device_version (optional)	int	4-bit device version of the endpoint.

### add\_endpoint\_response Parameters:

Parameter	Type	Description
endpoint_class	string	Name of the endpoint class that was instantiated.
endpoint_id	int	8-bit identifier that the endpoint was assigned.
profile_id	int	16-bit profile identifier of the endpoint.
device_id	int	16-bit device identifier of the endpoint, if it exists.
device_version	int	4-bit device version of the endpoint, if it exists.

## remove\_endpoint

Removes an endpoint with the given endpoint ID which has been previously added from endpoints.ini on startup or via the add\_endpoint command. Endpoints added in any other fashion cannot be removed by this command. Also removes the endpoint from endpoints.ini so that it will not be instantiated at startup.

Note: Any clusters on the given endpoint will also be removed when the endpoint is removed.

### remove\_endpoint Parameters:

Parameter	Type	Description
endpoint_id	int	8-bit identifier of the endpoint to be removed.

### remove\_endpoint\_response Parameters:

Parameter	Type	Description
endpoint_id	int	8-bit identifier of the endpoint that was removed.

## add\_cluster

Instantiates a cluster of the given class and adds it to the given endpoint. Optional parameters will override defaults of the cluster class. The cluster will also be added to clusters.ini so that on startup the cluster will be added to the given endpoint.

### add\_cluster Parameters:

Parameter	Type	Description
endpoint_id	int	8-bit identifier of the endpoint to which the cluster will be added.
cluster_class	string	Name of the cluster class to be instantiated.
cluster_id (optional)	int	16-bit identifier of the cluster. Required if cluster class does not provide a default.
server_or_client (optional)	int	If 0, the cluster will be a server cluster. If 1, the cluster will be a client cluster. Required if cluster class does not provide a default.
enable_APS_encryption (optional)	bool	If TRUE, all transmissions over this cluster will use and require APS-level encryption. If FALSE, this cluster will not use APS-level encryption.

### add\_cluster\_response Parameters:

Parameter	Type	Description
endpoint_id	int	8-bit identifier of the endpoint to which the cluster was added.
cluster_class	string	16-bit identifier of the cluster that was added.
cluster_id	int	16-bit identifier of the cluster.
server_or_client	int	Whether the cluster is a server (0) or client (1) cluster.
enable_APS_encryption	bool	Whether the cluster uses APS encryption or not.

### remove\_cluster

Removes a cluster of the given class from the given endpoint. If a class is not given, will remove a cluster with the given type (server or client) and ID from the endpoint. Only clusters which have been dynamically added from clusters.ini on startup or via the add\_cluster command may be removed in this fashion. Clusters added in any other fashion cannot be removed by this command. Also removes the cluster from clusters.ini so that it will not be added to the endpoint at startup.

### remove\_cluster Parameters:

Parameter	Type	Description
endpoint_id	int	8-bit identifier of the endpoint from which the cluster will be removed.
cluster_class (optional)	string	Name of the cluster class which is to be removed.
cluster_id (optional)	int	16-bit identifier of the cluster to be removed. Required if cluster class not given or cluster class does not define cluster ID.
server_or_client (optional)	int	If 0, the cluster to be removed is a server cluster. If 1, the cluster to be removed is a client cluster. Required if cluster class not given or cluster class does not define server_or_client.

### remove\_cluster\_response Parameters:

Parameter	Type	Description
endpoint_id	int	8-bit identifier of the endpoint from which the cluster was removed.
cluster_class (optional)	string	Class name of the removed cluster. Only present if given as a parameter to the remove_cluster command.
cluster_id	int	16-bit identifier of the cluster that was removed.
server_or_client	int	If 0, the removed cluster was a server cluster. If 1, the removed cluster was a client cluster.

## get\_version

Returns the version information of the given module. If no module is specified, returns the overall version of the firmware. Each module can be defined with a `__version__` and `__version_extended__` variable that can be set to any type. (See “Simple Parameter Types” on page 57 and “Complex Parameter Types” on page 58.) This command will return version information for the given module. If no module is given overall firmware version will be retrieved from Version.py.

### get\_version Parameters:

Parameter	Type	Description
module_name (optional)	string	The version information of the given module will be returned, defaults to Version module if not provided.

### get\_version\_response Parameters:

Parameter	Type	Description
module_name	string	Name of the module whose version was requested.
version	any	Version of the requested module. Can be any type depending on module definition. Typically a string such as "1.0.0".
version_extended	any	Extended version information of the requested module. Can be any type depending on module definition.

## help

Returns a listing of classes, methods and registry settings that are relevant to other RPC functionality. If no parameters are provided, then only a list of possible parameters to the help command is returned.

### help Parameters:

Parameter	Type	Description
registry (optional)	none	If provided, all registry entries will be listed.
methods (optional)	none	If provided, all RPC methods from instantiated interface classes will be listed.
interfaces (optional)	none	If provided, all known RPC interface classes will be listed.
endpoints (optional)	none	If provided, all known endpoint classes will be listed.
clusters (optional)	none	If provided, all known cluster classes will be listed.
records (optional)	none	If provided, all known record classes will be listed.
paths (optional)	none	If provided, all extra paths will be listed. This can include zip files.

## help\_response Parameters:

Parameter	Type	Description
registry (optional)	list	Included only if requested.  <b>item</b> - dict - Contains the information about the given registry entry. <b>name</b> - string - Name of the registry entry. <b>value</b> - int, bool, string or none - Current value of the registry entry. <b>description</b> - string - doc string of the registry entry.
methods (optional)	list	Included only if requested.  <b>item</b> - dict - Contains the information about the given RPC method. <b>name</b> - string - Name of the method. <b>module</b> - string - Name of the module where the method is defined. <b>description</b> - string - doc string of the method.
interfaces (optional)	list	Included only if requested.  <b>item</b> - dict - Contains the information about the given interface class. <b>name</b> - string - Name of the interface class. <b>module</b> - string - Name of the module where the interface class is defined. <b>description</b> - string - doc string of the interface class.
endpoints (optional)	list	Included only if requested.  <b>item</b> - dict - Contains the information about the given endpoint class. <b>name</b> - string - Name of the endpoint class. <b>module</b> - string - Name of the module where the endpoint class is defined. <b>description</b> - string - doc string of the endpoint class.
clusters (optional)	list	Included only if requested.  <b>item</b> - dict - Contains the information about the given cluster class. <b>name</b> - string - Name of the cluster class. <b>module</b> - string - Name of the module where the cluster class is defined. <b>description</b> - string - doc string of the cluster class.
records (optional)	list	Included only if requested.  <b>item</b> - dict - Contains the information about the given record class. <b>name</b> - string - Name of the record class. <b>module</b> - string - Name of the module where the record class is defined. <b>description</b> - string - doc string of the record class.
paths (optional)	list	Included only if requested. <b>item</b> - string - Contains the path that was dynamically added to the system

## **exit**

Terminates the program and associated processes. No parameters. No response.

## **message (response only)**

This is an unsolicited message sent by the gateway and typically indicates either operational status or errors.

Note: Only messages with severity greater than or equal to `RPC_General_Interface.debug_rpc_severity` will be sent.  
(See “`RPC_General_Interface.debug_rpc_severity`” on page 46)

### **message Parameters:**

Parameter	Type	Description
severity	int	Severity of the message. Can be 0 (debug), 1 (warning), or 2 (error).
description (optional)	string	Contents of the message.
request_identifier (optional)	can be any type	Present if this message corresponds to a previous RPC request that included a request_identifier.
(various) (optional)	can be any type	Messages may include other parameters to provide more information.

## **get\_time**

Gets the current universal coordinate time from the gateway. This command takes no parameters.

### **get\_time\_response Parameters:**

Parameter	Type	Description
UTC_1970	int	Number of seconds since Jan. 1, 1970, universal coordinate time
UTC_2000	int	Number of seconds since Jan. 1, 2000, universal coordinate time

## **set\_time**

Sets the time on the gateway using universal coordinate time. The time can be set using 1970 or 2000 as the EPOCH.



**set\_time Parameters:**

Parameter	Type	Description
UTC_1970 (optional)	int	Set the number of seconds since Jan. 1, 1970, universal coordinate time
UTC_2000 (optional)	int	Set the number of seconds since Jan. 1, 2000, universal coordinate time

**set\_time\_response Parameters:**

Parameter	Type	Description
UTC_1970	int	Number of seconds since Jan. 1, 1970, universal coordinate time
UTC_2000	int	Number of seconds since Jan. 1, 2000, universal coordinate time

**resync\_time**

Forces a resynchronization of time with an NTP or ZCL time server as appropriate. This RPC command takes no parameters.

**set\_time\_response - no parameters**



## ZigBee Requests/Responses

### add\_device

Adds the device to the list of known devices and to devices.ini. Additionally, if the XBee on the gateway is a coordinator and a link key or installation code is provided, the link key for the device will be added to the link key table of the XBee. This will allow the remote device to join the network. An optional join time can be specified which will temporarily enable node joining on the network.

#### add\_device Parameters:

Parameter	Type	Description
device_address	MAC	64-bit extended address of the remote device to be added.
join_time (optional)	int	Number of seconds for which node joining will be enabled on the network once the device has been added locally. If not provided, node joining behavior will not be affected.
link_key (optional)	int, base16 or base64.	The link key of the remote device.
installation_code (optional)	string, base16 or base64.	The installation code of the remote device. If link_key is also provided, installation_code will be ignored. If provided as a string any non hex characters will be stripped before generating the link key to allow formatting characters to be included in installation_code.

#### add\_device\_response Parameters:

Parameter	Type	Description
device_address	MAC	64-bit extended address of the remote device that was added.
join_time (optional)	int	Only present if given as a parameter to the add_device request.

### remove\_device

Removes a device from the list of known devices and from endpoints.ini. Additionally, if the XBee on the gateway is a coordinator, the link key for that device will be removed from the link key table of the XBee. Once the device is removed a ZDO Mgmt\_Leave\_req is sent to the device.

### remove\_device Parameters:

Parameter	Type	Description
device_address	MAC	64-bit extended address of the remote device to be removed.

### remove\_device\_response Parameters:

Parameter	Type	Description
device_address	MAC	64-bit extended address of the remote device that was removed.

## enable\_joining

Enables node joining on the network for the specified number of seconds. If 0, node joining will immediately be disabled.

### enable\_joining Parameters:

Parameter	Type	Description
join_time	int	Number of seconds for which node joining will be enabled on the network.

### enable\_joining\_response Parameters:

Parameter	Type	Description
join_time	int	Number of seconds for which node joining will be enabled on the network.

## refresh\_device\_information

Initiates a periodic refresh immediately. This command can also initiate a full refresh for existing devices. A full refresh updates all known information such as endpoints, clusters, power and node descriptors, etc. (See “Periodic Refresh” on page 43)

### refresh\_device\_information Parameters:

Parameter	Type	Description
device_address (optional)	MAC	If provided, then will only refresh specified device. Defaults to a broadcast for all devices.
full_refresh (optional)	bool	If TRUE, then perform a full refresh. If FALSE, then perform a standard periodic refresh. Defaults to FALSE.

## get\_device\_information

Returns all known information about devices on the network.

### get\_device\_information Parameters:

Parameter	Type	Description
device_address (optional)	MAC	64-bit address of the device about which to return information. If not provided, the device information for every known device will be returned.

### get\_device\_information\_response Parameters:

Parameter	Type	Description
record_list	list	Each item in the list corresponds to a device on the network. <b>item</b> - ZDODeviceRecord (See “ZDODeviceRecord” on page 116)

## xbee\_AT

The XBee radio on the gateway provides a set of AT commands to control register values and issue commands. The xbee\_AT request provides direct and unmanaged access to these commands. It is possible for certain AT commands to interfere with normal operation of the gateway. For example, changing the baud rate may prevent the gateway from sending and receiving messages on the Smart Energy network. Except as instructed in this or other Smart Energy gateway documentation this command should only be used by advanced users. The XBee SE Manual has further documentation on XBee AT commands (see “Online Documentation” on page 7).

### xbee\_AT Parameters:

Parameter	Type	Description
command	string	Case insensitive two character XBee AT command code.
value (optional)	int, base16, or base64	When included may be used to set a register value or otherwise provide command parameters. Do not include to get a register value in the response or when sending a command with no parameters.

### xbee\_AT\_response Parameters:

Parameter	Type	Description
command	string	Same as sent in the original request.
value (optional)	int	If not included in the original request then value will be the XBee's response parameter, if available. Otherwise value will be the same as sent in the original request.

## bind

Sends a ZDO bind request to a device.

### bind Parameters:

Parameter	Type	Description
destination_address	MAC	64-bit extended address of the device to which to send the ZDO bind request
destination_endpoint_id	int	8-bit identifier of the endpoint on the remote device on which the target cluster exists
source_endpoint_id	int	8-bit identifier of the endpoint on the local device on which the target cluster exists
cluster_id	int	16-bit identifier of the cluster to be bound by the bind request

### bind\_request Parameters:

Parameter	Type	Description
status	int	Indicates the success or failure of the bind request using standard ZDO status values. Can be success (0x00), not supported (0xAA) or other
destination_address	MAC	64-bit extended address of the device to which the ZDO bind request was sent
destination_endpoint_id	int	8-bit identifier of the endpoint on the remote device on which the target cluster exists
source_endpoint_id	int	8-bit identifier of the endpoint on the local device on which the target cluster exists
cluster_id	int	16-bit identifier of the cluster to be bound by the bind request

## unbind

Sends a ZDO unbind request to a device. Unbind requests remove previous bind requests.

### unbind Parameters:

Parameter	Type	Description
destination_address	MAC	64-bit extended address of the device to which to send the ZDO unbind request
destination_endpoint_id	int	8-bit identifier of the endpoint on the remote device on which the target cluster exists
source_endpoint_id	int	8-bit identifier of the endpoint on the local device on which the target cluster exists
cluster_id	int	16-bit identifier of the cluster to be unbound by the unbind request

### unbind\_response Parameters:

Parameter	Type	Description
status	int	Indicates the success or failure of the unbind request using standard ZDO status values. Can be success (0x00), not supported (0xAA), or other
destination_address	MAC	64-bit extended address of the device to which the ZDO bind request was sent
destination_endpoint_id	int	8-bit identifier of the endpoint on the remote device on which the target cluster exists
source_endpoint_id	int	8-bit identifier of the endpoint on the local device on which the target cluster exists
cluster_id	int	16-bit identifier of the cluster to be bound by the bind request

### configure\_zigbee\_network

Sets the parameters to join or form a network. Sending this command will cause the gateway to reform or leave the current network it is joined to.

### configure\_zigbee\_network Parameters:

Parameter	Type	Description
extended_pan_id (optional)	int	<b>If gateway is a coordinator</b> - Extended PAN ID of the network to form. If set to zero, network will be formed on a random extended PAN ID. <b>If gateway is a router</b> - Extended PAN ID of the network to join. If set to zero, the gateway will not filter join attempts on extended PAN ID.
channel_mask (optional)	int	16-bit bitmask of the channels to use for joining or forming a network. Bits 0 - 16 map directly to channels 0x0B - 0x1A.
link_key (optional)	int, base16	128-bit link key used to join a network.
installation_code (optional)	int, base16, string	48-, 64-, 96-, or 128-bit installation code hashed to generate a 128-bit link key to join a network. Must include CRC as part of installation code.
join_interval (optional)	int	Interval between join attempts for router. This parameter will also set the registry_confirmation parameter ZDO_Device_Manager.join_interval (see Registry on page 45).



### **configure\_zigbee\_network\_response Parameters:**

Parameter	Type	Description
extended_pan_id	int	Extended PAN ID set on the gateway.
channel_mask	int	16-bit bitmask of the channels to use for joining or forming a network. Bits 0 - 16 map directly to channels 0x0B - 0x1A.
join_interval	int	Interval between join attempts for router. This parameter will also set the registry_confirmation parameter ZDO_Device_Manager.join_interval.

### **leave\_network**

Will cause a router gateway to leave the network (no effect on the coordinator). This command takes no parameters.

**leave\_network\_response** - has no parameters.

### **get\_zigbee\_network\_status**

Gives the status of the gateway on the ZigBee network. The status is given as both a human-readable string and an enumerated status value. This command takes no parameters

### **get\_zigbee\_network\_status\_response Parameters:**

Parameter	Type	Description
status	int	8-bit enumerated status value. Corresponds to the Associate Indication response given from an XBee ATAI command. 0x00 indicates the gateway is successfully joined to a network.
status_description	string	Human-readable string version of the status.
extended_pan_id	int, base16	64-bit operating extended PAN ID of the ZigBee network of the gateway.
pan_id	int	16-bit operating PAN ID of the ZigBee network of the gateway.
channel	int	The channel of the ZigBee network of the gateway.
addr_short	MAC	16-bit network address of the XBee in the gateway
addr_extended	MAC	64-bit Extended Unique Identifier (EUI aka MAC address) of the XBee in the gateway
configured_extended_pan_id	int	64-bit configured extended PAN ID of the ZigBee network on the gateway. 0x0 has special meaning. On a coordinator, 0x0 forms the network on any PAN ID. On a router, 0x0 will try to join any network.

## ZCL Requests/Responses

In order to perform a ZCL command on a remote server cluster, a corresponding client cluster must exist locally. In order to perform a ZCL command on a remote client cluster, a corresponding server cluster must exist locally. If necessary, the `add_cluster` command can be used to create a ZCL cluster of the appropriate type. (See “`add_cluster`” on page 67)

Note: Some local ZCL requests do not have this limitation.

### `read_attributes`

Reads ZCL attributes from the given cluster on the given device and endpoint.

#### `read_attributes` Parameters:

Parameter	Type	Description
<code>cluster_id</code>	int	16-bit identifier of the cluster to which the ZCL command will be sent.
<code>server_or_client</code>	int	Target cluster is a server (0) or client (1) cluster.
<code>destination_endpoint_id</code>	int	8-bit identifier of the endpoint on which the target cluster exists.
<code>source_endpoint_id</code> (optional)	int	8-bit identifier of the endpoint from which the ZCL command will be sent. Only required for remote commands.
<code>destination_address</code> (optional)	MAC	64-bit extended address of the device to send the ZCL command to. If not provided, the request will be to the local device.
<code>manufacturer_code</code> (optional)	int	16-bit manufacturer code of the attributes in the record list.
<code>record_list</code>	list	List of records which will make up the payload of the ZCL Read Attributes Command.  <b>Item</b> - ReadAttributeRecord. (See “ReadAttributeRecord” on page 119)

#### `read_attributes_response` Parameters:

Parameter	Type	Description
<code>cluster_id</code>	int	16-bit identifier of the cluster from which the response was sent.
<code>server_or_client</code>	int	Indicates whether the cluster from which the response was sent was a server (0) or client (1) cluster.
<code>profile_id</code>	int	16-bit profile identifier of the endpoint from which the response was sent.
<code>source_endpoint_id</code>	int	8-bit identifier of the endpoint from which the response was sent.
<code>destination_endpoint_id</code> (optional)	int	8-bit identifier of the endpoint to which the response was sent. Only included if the request was not local.
<code>source_address</code> (optional)	MAC	64-bit extended address of the device from which the response was sent. Only included if the request was not local.



Parameter	Type	Description
manufacturer_code (optional)	int	16-bit manufacturer code of the attributes in the record list.
record_list	list	List of records which made up the payload of the ZCL Read Attributes Response Command. <b>item</b> - ReadAttributeStatusRecord (See “ReadAttributeStatusRecord” on page 120.)

## write\_attributes

Writes to ZCL attributes on the given cluster on the given device and endpoint. Writes may be normal or undivided. If any attribute in an undivided write cannot be written then none of the attributes will be written. In a normal write all attributes are written individually with success or failure returned for each.

### write\_attributes Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the cluster to which the ZCL command will be sent.
server_or_client	int	Indicates whether the target cluster is a server (0) or client (1) cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on which the target cluster exists.
source_endpoint_id (optional)	int	8-bit identifier of the endpoint from which the ZCL command will be sent. Only required for remote commands.
destination_address (optional)	MAC	64-bit extended address of the device to send the ZCL command to. If not provided, the command will be to the local device.
command_identifier (optional)	int	8-bit ZCL command identifier to specify Write Attributes Command (0x02) or Write Attributes Undivided Command (0x05). Defaults to Write Attributes Command.
manufacturer_code (optional)	int	If included, this is the 16-bit manufacturer code of the attributes in the record list.
record_list	list	List of records which will make up the payload of the ZCL Write Attributes Command. <b>item</b> - WriteAttributeRecord (See “WriteAttributeRecord” on page 119)

### write\_attributes\_response Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the target cluster from which the response was sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.
profile_id	int	16-bit profile identifier of the endpoint from which the response was sent.
source_endpoint_id	int	8-bit identifier of the endpoint from which the response was sent.

Parameter	Type	Description
destination_endpoint_id (optional)	int	8-bit identifier of the endpoint to which the response was sent.
source_address (optional)	MAC	64-bit extended address of the device from which the response was sent.
manufacturer_code (optional)	int	16-bit manufacturer code of the attributes in the record list.
record_list	list	List of records which made up the payload of the ZCL Write Attributes Response Command. <b>item</b> - WriteAttributeResponseRecord. (See “WriteAttributeResponseRecord” on page 120)

## discover\_attributes

Discovers all ZCL attributes on the target cluster.

### discover\_attributes Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the target cluster to which the ZCL command will be sent.
server_or_client	int	Target cluster is a server (0) or client (1).
destination_endpoint_id	int	8-bit identifier of the endpoint on which the target cluster exists.
source_endpoint_id (optional)	int	8-bit identifier of the endpoint from which the ZCL command will be sent. Required for remote requests.
destination_address (optional)	MAC	64-bit extended address of the device to send the ZCL command to.
manufacturer_code (optional)	int	If included, the 16-bit manufacturer code of the attributes to be discovered. Defaults to no manufacturer code.

### discover\_attributes\_response Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the target cluster from which the response was sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.
profile_id	int	16-bit profile identifier of the endpoint from which the response was sent.
source_endpoint_id	int	8-bit identifier of the endpoint from which the response was sent.

Parameter	Type	Description
destination_endpoint_id (optional)	int	8-bit identifier of the endpoint to which the response was sent.
source_address (optional)	MAC	64-bit extended address of the device from which the response was sent.
manufacturer_code (optional)	int	16-bit manufacturer code of the attributes in the record list.
record_list	list	List of records which made up the payload of the ZCL Discover Attributes Response Command.  <b>item</b> - AttributeInformationRecord. (See "AttributeInformationRecord" on page 123)

## start\_receiving\_reports

Configures the target cluster to begin sending ZCL Report Attributes Commands for the specified attributes to the local device. Can also be used to enable pseudo reporting on the gateway, where the gateway sends Read Attribute Commands to the remote device to simulate enabling reporting on the device.

### start\_receiving\_reports Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the target cluster to which the ZCL command will be sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on which the target cluster exists.
pseudo_reporting (optional)	bool	If TRUE, will use pseudo reporting instead of ZCL reporting. If FALSE, will use ZCL reporting. If omitted or set to none, will try to use ZCL reporting. If that fails or isn't supported, will use pseudo reporting.
source_endpoint_id (optional)	int	8-bit identifier of the endpoint from which the ZCL command will be sent. Required for remote requests.
destination_address (optional)	MAC	64-bit extended address of the device to send the ZCL command to. Defaults to the local device.
manufacturer_code (optional)	int	If included, the 16-bit manufacturer code of the attributes in the record list. Defaults to no manufacturer code.
record_list	list	List of records will make up the payload of the ZCL Configure Reporting Command.  <b>item</b> - AttributeReportingConfigurationRecord (See "AttributeReportingConfigurationRecord" on page 120)

### start\_receiving\_reports\_response Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the target cluster from which the response was sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.
profile_id	int	16-bit profile identifier of the endpoint from which the response was sent.
source_endpoint_id	int	8-bit identifier of the endpoint from which the response was sent.
destination_endpoint_id (optional)	int	8-bit identifier of the endpoint to which the response was sent.
source_address (optional)	MAC	64-bit extended address of the device from which the response was sent.
manufacturer_code (optional)	int	16-bit manufacturer code of the attributes in the record list.
record_list	list	List of records which made up the payload of the ZCL Configure Reporting Response Command.  <b>item</b> - AttributeReportingConfigurationResponseRecord (See "AttributeReportingConfigurationResponseRecord" on page 121)

### attribute\_report (response only)

Attribute reports which arrive from a remote device will generate an attribute\_report RPC response.

Note: Only one attribute will be included in each attribute\_report, even if multiple attributes are configured for reporting at once.

### attribute\_report Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the cluster from which the report was sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.
profile_id	int	16-bit profile identifier of the endpoint from which the report was sent.
source_endpoint_id	int	8-bit identifier of the endpoint from which the report was sent.
destination_endpoint_id (optional)	int	8-bit identifier of the endpoint to which the report was sent.
source_address (optional)	MAC	64-bit extended address of the device from which the report was sent.

Parameter	Type	Description
manufacturer_code (optional)	int	16-bit manufacturer code of the attribute in the record.
record	AttributeReportRecord	A single attribute report extracted from the payload of a ZCL Report Attributes Command. (See “AttributeReportRecord” on page 121)

## stop\_receiving\_reports

Configures the target cluster to stop sending ZCL Report Attributes Commands for the specified attributes to the local device.

### stop\_receiving\_reports Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the target cluster to which the ZCL command will be sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on which the target cluster exists.
source_endpoint_id	int	8-bit identifier of the endpoint from which the ZCL command will be sent.
destination_address	MAC	64-bit extended address of the device to send the ZCL command to.
manufacturer_code (optional)	int	If included, this is the 16-bit manufacturer code of the attributes in the record list. Defaults to no manufacturer code.
record_list	list	List of records which will make up the payload of the ZCL Configure Reporting Command.  <b>item</b> - StopReportingRecord (See “StopReportingRecord” on page 122)

### stop\_receiving\_reports\_response Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the target cluster from which the response was sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.
profile_id	int	16-bit profile identifier of the endpoint from which the response was sent.
source_endpoint_id	int	8-bit identifier of the endpoint from which the response was sent.
destination_endpoint_id	int	8-bit identifier of the endpoint to which the response was sent.

Parameter	Type	Description
source_address	MAC	64-bit extended address of the device from which the response was sent.
manufacturer_code (optional)	int	16-bit manufacturer code of the attributes in the record list.
record_list	list	List of records derived from the payload of the ZCL Configure Reporting Response Command.  <b>item</b> - StopReportingStatusRecord (See “StopReportingStatusRecord” on page 122)

## stop\_sending\_reports

Configures the local device to stop sending specified ZCL Report Attributes Commands to the destination device. Also sets the destination device's report timeout to 0. It is currently not possible within ZCL to notify a remote device that reports are no longer being generated.

### stop\_sending\_reports Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the target cluster to which the ZCL command will be sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on which the target cluster exists.
source_endpoint_id	int	8-bit identifier of the endpoint from which the ZCL command will be sent.
destination_address	MAC	64-bit extended address of the device to send the ZCL command to.
manufacturer_code (optional)	int	If included, the 16-bit manufacturer code of the attributes in the record list. Defaults to no manufacturer code.
record_list	list	List of records which will make the payload of the ZCL Configure Reporting Command.  <b>item</b> - StopReportingRecord (See “StopReportingRecord” on page 122)

### stop\_sending\_reports\_response Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the target cluster from which the response was sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.
profile_id	int	16-bit identifier of the profile of the endpoint from which the response was sent.

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint from which the response was sent.
destination_endpoint_id	int	8-bit identifier of the endpoint to which the response was sent.
source_address	MAC	64-bit extended address of the device from which the response was sent.
manufacturer_code (optional)	int	16-bit manufacturer code of the attributes in the record list.
record_list	list	List of records derived from the payload of the ZCL Configure Reporting Response Command.  <b>item</b> - StopReportingStatusRecord (See "StopReportingStatusRecord" on page 122)

## get\_local\_reporting\_configurations

Returns all of the reporting configurations of the gateway. This includes configurations for gateway reporting to other devices and other devices reporting to the gateway. This command takes no parameters.

### get\_local\_reporting\_configurations\_response Parameters:

Parameter	Type	Description
record_list	list	List of records which specify the local reporting configurations on the gateway.  <b>item</b> - LocalReportingConfigurationRecord (see "LocalReportingConfigurationRecord" on page 125)

## read\_reporting\_configuration

Retrieves the ZCL Attribute reporting configuration records for the specified attribute reports.

### read\_reporting\_configuration Parameters:

Parameter	Type	Description
remote_configuration (optional)	bool	If TRUE, the configuration is to be read from the remote device's destination_endpoint. If FALSE, the configuration is to be read from the local device's source_endpoint. Defaults to FALSE
cluster_id	int	16-bit identifier of the target cluster to which the ZCL command will be sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.

Parameter	Type	Description
reporting_direction	int	Configuration to be read is for reports being sent from (0) or sent to (1) the target cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on which the target cluster exists.
source_endpoint_id (optional)	int	8-bit identifier of the endpoint from which the ZCL command will be sent.
destination_address	MAC	64-bit extended address of the device hosting the destination endpoint.
manufacturer_code (optional)	int	If included, the 16-bit manufacturer code of the attributes in the record list. Defaults to no manufacturer code.
record_list	list	List of records which will make up the payload of the ZCL Read Reporting Configuration Response Command  <b>item</b> - ReadReportingConfigurationRecord (See "ReadReportingConfigurationRecord" on page 122)

#### read\_reporting\_configuration\_response Parameters:

Parameter	Type	Description
remote_configuration (optional)	bool	If TRUE, the configuration was read from the remote device's source_endpoint (destination_endpoint in request). If FALSE, the configuration was read from the local device's destination_endpoint (source_endpoint in request).
cluster_id	int	16-bit identifier of the cluster from which the response was sent.
server_or_client	int	Cluster from which the response was sent is a server (0) or client (1) cluster.
profile_id	int	16-bit profile identifier of the endpoint from which the response was sent.
source_endpoint_id	int	8-bit identifier of the endpoint from which the response was sent.
destination_endpoint_id (optional)	int	8-bit identifier of the endpoint to which the response was sent.
source_address	MAC	64-bit extended address of the device hosting the source endpoint (destination endpoint in request).
manufacturer_code (optional)	int	If included, the 16-bit manufacturer code of the attributes in the record list.
record_list	list	List of records which made up the payload of the ZCL Read Reporting Configuration Response Command.  <b>item</b> - ReadReportingConfigurationResponseRecord (See "ReadReportingConfigurationResponseRecord" on page 122)



## identify

Instructs the target device to begin self-identification. When the target device is a gateway, it will rapidly blink its associate LED for self-identification.

### identify Parameters:

Parameter	Type	Description
destination_endpoint_id	int	8-bit identifier of an endpoint hosting the Identify server cluster (0x0003).
source_endpoint_id (optional)	int	8-bit identifier of the endpoint from which the ZCL request will be sent. Required for remote requests.
destination_address (optional)	MAC	64-bit extended address of the device to send the ZCL request to. Defaults to the local device.
identify_time	int	Number of seconds for which identification should occur.

### identify\_response Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the cluster from which the response was sent.
profile_id	int	16-bit identifier of the profile of the endpoint from which the response was sent.
source_endpoint_id	int	8-bit identifier of the endpoint from which the response was sent.
destination_endpoint_id (optional)	int	8-bit identifier of the endpoint to which the response was sent.
source_address (optional)	MAC	64-bit extended address of the device from which the response was sent.
identify_time	int	Number of seconds for which identification should occur.

## send\_ZCL

Generates and sends a ZCL command with the given header information and payload to a remote device. This command supports pass-through of custom ZCL commands which do not have an associated interface method defined.

## send\_ZCL Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the target cluster to which the request will be sent.
server_or_client	int	Target cluster is a server (0) or client (1) cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on which the target cluster exists.
source_endpoint_id	int	8-bit identifier of the endpoint from which the ZCL request will be sent.
destination_address	MAC	64-bit extended address of the device to send the ZCL request to.
command_identifier	int	8-bit command identifier of the ZCL request.
command_type (optional)	int	Command is a general ZCL (0) or cluster-specific (1) command. Defaults to cluster-specific (1).
manufacturer_code (optional)	int	If included, the 16-bit manufacturer code of the attributes for general ZCL commands or the manufacturer code of the command identifier for cluster-specific commands.
response_command_identifier (optional)	int	8-bit command identifier of the ZCL response. If provided this will be used to match a ZCL response to this request.
response_command_type (optional)	int	Response command will be a general ZCL (0) or cluster-specific (1) command. Defaults to the value of command_type.
enable_aps_encryption (optional)	bool	If TRUE, the ZCL request will be sent with APS encryption enabled. If FALSE the ZCL request will be sent with APS encryption disabled. Defaults to the sending cluster's configuration.
payload	base16, base64, record or list of records.	Payload of the ZCL command.

## send\_ZCL\_response Parameters:

Parameter	Type	Description
cluster_id	int	16-bit identifier of the cluster from which the response was sent.
server_or_client	int	Target cluster from which the response was sent is a server (0) or client (1) cluster.
profile_id	int	16-bit profile identifier of the endpoint from which the response was sent.
source_endpoint_id	int	8-bit identifier of the endpoint from which the response was sent.
destination_endpoint_id	int	8-bit identifier of the endpoint to which the response was sent.
source_address	MAC	64-bit extended address of the device from which the response was sent.
command_identifier	int	8-bit command identifier of the ZCL response.
command_type	int	Response command was a general ZCL (0) or cluster-specific (1) command.
manufacturer_code (optional)	int	If included, the 16-bit manufacturer code of the response command.
enable_aps_encryption (optional)	bool	If TRUE, the response command was APS encrypted. If FALSE, the command was not APS encrypted. Only included if the original request provided this parameter.
payload	base16 or base64	Payload of the ZCL response. It will be of type base64 if the original request used base64 for its payload. Otherwise it will be of type base16.



## SE Requests/Responses

### *Demand Response / Load Control (DRLC) Commands - Common*

#### **get\_DRLC\_events**

Returns the gateway's list of valid DRLC events. This command can be used on a DRLC server or client cluster.

#### **get\_DRLC\_events Parameters:**

Parameter	Type	Description
endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the DRLC cluster. Defaults to the standard SE endpoint (0x5E).
server_or_client (optional)	int	Return events from local server (0) or client (1) cluster. If not specified, will search endpoint for DRLC cluster.

#### **get\_DRLC\_events\_response Parameters:**

Parameter	Type	Description
endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the DRLC server cluster.
events_record_list	list	List of valid events.  <b>item</b> - LoadControlEventRecord - Contains information about the event. (See "LoadControlEventRecord" on page 125)

### *Demand Response / Load Control (DRLC) Commands - Server*

#### **create\_DRLC\_event**

Adds a DRLC event to the gateway's list of events. A Load Control Event command (0x00) will be immediately sent to all DRLC client clusters hosted on an active device. This command will be sent from the specified endpoint hosting a DRLC server cluster on the local device.

### create\_DRLC\_event Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the DRLC server cluster. Defaults to the standard SE endpoint (0x5E).
record	LoadControlEventRecord	Contains information about the event to add. (See "LoadControlEventRecord" on page 125).

### create\_DRLC\_event\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the DRLC server cluster.
cluster_id	int	16-bit identifier of the cluster to which the DRLC event was sent.
status	int	Indicates the local success or failure of adding the event using standard ZCL status values. Can be success (0x00), invalid field (0x85) or duplicate exists (0x8A).
record	LoadControlEventRecord	Record containing information about the event that was added. (See "LoadControlEventRecord" on page 125).
device_list	list	List of all devices to which the event was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the event was sent.

### cancel\_DRLC\_event

Removes a DRLC event from the gateway's list of events. A Cancel Load Control Event command (0x01) will be immediately sent to all DRLC client clusters hosted on an active device. This command will be sent from the specified endpoint hosting a DRLC server cluster on the local device.

### cancel\_DRLC\_event Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the DRLC server cluster. Defaults to the standard SE endpoint (0x5E).
record	CancelLoadControlEventRecord	Contains information about the cancel. (See "CancelLoadControlEventRecord" on page 125)

### cancel\_DRLC\_event\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the DRLC server cluster.
cluster_id	int	16-bit identifier of the cluster to which the DRLC event was sent.
status	int	Indicates the local success or failure of cancelling the event using standard ZCL status values. A cancel should never fail locally so this field should always contain success (0x00).
record	CancelLoadControlEventRecord	Contains information about the cancel. (See "CancelLoadControlEventRecord" on page 125)
device_list	list	List of all devices to which the cancel was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the cancel was sent.

## cancel\_all\_DRLC\_events

Removes all DRLC events from the gateway's list of events. A DRLC Cancel All Load Control Events command (0x02) will be immediately sent to all client clusters hosted on an active device. This command will be sent from the specified endpoint hosting a DRLC server cluster on the local device.

### cancel\_all\_DRLC\_events Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the DRLC server cluster. Defaults to the standard SE endpoint (0x5E).
record	CancelAllLoadControlEventsRecord	Contains information about the cancel. (See "CancelAllLoadControlEventsRecord" on page 126)

### cancel\_all\_DRLC\_events\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the DRLC server cluster.
cluster_id	int	16-bit identifier of the cluster to which the DRLC event was sent.
status	int	Indicates the local success or failure of canceling all events using standard ZCL status values. A cancel should never fail locally so this field should always contain success (0x00).
record	CancelAllLoadControlEventsRecord	Contains information about the cancel. (See "CancelAllLoadControlEventsRecord" on page 126)
device_list	list	List of all devices to which the cancel was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the cancel was sent.

## received\_report\_event\_status (response only)

This RPC response is generated when a DRLC server cluster hosted on the local device receives a DRLC Report Event Status command (0x00).

### received\_report\_event\_status Parameters:

Parameter	Type	Description
source_address	MAC	64-bit extended address of the device hosting the DRLC client.
source_endpoint_id	int	8-bit identifier of the endpoint on the remote device hosting the DRLC client.
record	ReportEventStatusRecord	Describes the status of the event. (See "ReportEventStatusRecord" on page 126)
cluster_id	int	16-bit identifier of the DRLC cluster.
destination_endpoint_id	int	8-bit identifier of the local endpoint on which the DRLC cluster exists.

## Client

### get\_scheduled\_DRLC\_events

Get server to resend scheduled DRLC events. Client sends a get scheduled DRLC events command (0x01) to each of the active DRLC servers. This request can specify a start time and maximum number of events for the server to send.

### get\_scheduled\_DRLC\_events Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the DRLC client cluster. Defaults to the standard SE endpoint (0x5E).
record (optional)	GetScheduledEventsRecord	Specifies the start time and number of events to get from the DRLC server cluster. Defaults to getting all events. (See GetScheduledEventsRecord" on page 127.)

### get\_scheduled\_DRLC\_events\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the DRLC client cluster.
cluster_id	int	16-bit identifier of the cluster from which the DRLC event was sent.



Parameter	Type	Description
record (optional)	GetScheduledEventsRecord	Returns the start time and number of events requested from the DRLC server cluster. Only included in the response if record was included in the original request. (See "GetScheduledEventsRecord" on page 127.)
device_list	list	List of all devices to which the message was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the cancel was sent.

## clear\_DRLC\_events

Removes all DRLC events from the gateway's DRLC client cluster's list of events. Events are still active on the server and may be resent or requested to be resent by the client using the get\_scheduled\_DRLC\_events command.

### clear\_DRLC\_events Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the DRLC client cluster. Defaults to the standard SE endpoint (0x5E).

### clear\_DRLC\_events\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the DRLC client cluster.
cluster_id	int	16-bit identifier of the DRLC cluster.



## received\_DRLC\_event (response only)

This RPC response message is generated when a DRLC client cluster hosted on the local device received a Load Control Event command (0x00). The event will be added to the DRLC client.

### received\_DRLC\_event Parameters:

Parameter	Type	Description
status	int	Indicates the success or failure of receiving the DRLC event. Can be success (0x00), invalid field (0x85), or duplicate exists (0x8A).
source_address	MAC	64-bit extended address of the device hosting the DRLC server cluster.
source_endpoint_id	int	8-bit identifier of the endpoint on the remote device hosting the DRLC server cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on the gateway that hosts the DRLC client cluster.
cluster_id	int	16-bit identifier of the DRLC cluster.
record	LoadControlEventRecord	Contains information about the received event. (See "LoadControlEventRecord" on page 126.)

## received\_DRLC\_cancel\_event (response only)

This RPC response message is generated when a DRLC client cluster hosted on the local device received a Cancel Load Control Event command (0x01).

### received\_DRLC\_cancel\_event Parameters:

Parameter	Type	Description
status	int	Indicates the success or failure of receiving the DRLC cancel event. Can be success (0x00), invalid field (0x85), or duplicate exists (0x8B).
source_address	MAC	64-bit extended address of the device hosting the DRLC server cluster.
source_endpoint_id	int	8-bit identifier of the endpoint on the remote device hosting the DRLC server cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on the gateway that hosts the DRLC client cluster.

Parameter	Type	Description
cluster_id	int	16-bit identifier of the DRLC cluster.
record	CancelLoadControlEventRecord	Contains information about the received event. (See “CancelLoadControlEventRecord” on page 126.)

## received\_DRLC\_cancel\_all\_events (response only)

This RPC response message is generated when a DRLC client cluster hosted on the local device received a Cancel All Load Control Events command (0x02).

### received\_DRLC\_cancel\_all\_events Parameters:

Parameter	Type	Description
status	int	Indicates the success or failure of receiving the DRLC cancel all event. Can be success (0x00) or invalid field (0x85).
source_address	MAC	64-bit extended address of the device hosting the DRLC server cluster.
source_endpoint_id	int	8-bit identifier of the endpoint on the remote device hosting the DRLC server cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on the gateway that hosts the DRLC client cluster.
cluster_id	int	16-bit identifier of the DRLC cluster.
record	CancelAllLoadControlEventsRecord	Contains information about the received event. (See “CancelAllLoadControlEventsRecord” on page 127.)

## updated\_active\_DRLC\_events (response only)

The DRLC client cluster keeps a list of currently active DRLC events. When this list changes, the client generates an updated\_active\_DRLC\_events. This can occur when an event starts, stops, or is canceled. An empty list of events will be returned when there are no active events.

### updated\_active\_DRLC\_events Parameters:

Parameter	Type	Description
destination_endpoint_id	int	8-bit identifier of the endpoint on the gateway that hosts the DRLC client cluster.
cluster_id	int	16-bit identifier of the DRLC cluster.
record_list	list	List of active DRLC events. List will be empty if there are no active DRLC events. <b>item</b> - LoadControlEventRecord (See "LoadControlEventRecord" on page 126)



## ***Messaging Commands - Common***

### **get\_message\_events**

Returns the gateway's list of valid message events. This command can be used on a message server or client cluster.

#### **get\_message\_events Parameters:**

Parameter	Type	Description
endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Messaging cluster. Defaults to the standard SE endpoint (0x5E).

#### **get\_message\_events\_response Parameters:**

Parameter	Type	Description
endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Messaging cluster.
events_record_list	list	List of valid events.  <b>item</b> - DisplayMessageRecord - Contains information about the message. (See "DisplayMessageRecord" on page 127)

## ***Messaging Commands - Server***

### **create\_message\_event**

Add a message event to the gateway's list of events. A Display Message command (0x00) will be immediately sent to all Messaging client clusters hosted on an active device. This command will be sent from the specified endpoint hosting a Messaging server cluster on the local device.

### create\_message\_event Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Messaging server cluster. Defaults to the standard SE endpoint (0x5E).
record	DisplayMessageRecord	Contains information about the event to add. (See "DisplayMessageRecord" on page 127).

### create\_message\_event\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Messaging server cluster.
cluster_id	int	16-bit identifier of the cluster to which the Messaging command was sent.
status	int	Indicates the local success or failure of adding the event using standard ZCL status values. Can be success (0x00), invalid field (0x85) or duplicate exists (0x8A).
record	DisplayMessageRecord	Contains information about the event that was added. (See "DisplayMessageRecord" on page 127).
device_list	list	List of all devices to which the event was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the event was sent.

## cancel\_message\_event

Removes a message event from the gateway's list of events. A Cancel Message command (0x01) will be immediately sent to all Messaging client clusters hosted on an active device. This command will be sent from the specified endpoint hosting a Messaging server cluster on the local device.

### cancel\_message\_event Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Messaging server cluster. Defaults to the standard SE endpoint (0x5E).
record	CancelMessageRecord	Contains information about the cancel. (See "CancelMessageRecord" on page 127).

### cancel\_message\_event\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Messaging server cluster.
cluster_id	int	16-bit identifier of the cluster to which the Messaging command was sent
status	int	Indicates the local success or failure of cancelling the event using standard ZCL status values. A cancel should never fail locally so this field should always contain success (0x00).
record	CancelMessageRecord	Contains information about the cancel. (See "CancelMessageRecord" on page 127).
device_list	list	List of all devices to which the cancel was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the cancel was sent.

### cancel\_all\_message\_events

Removes all message events from the gateway's list of events. The Messaging cluster does not define a Cancel All Messages command so each message must be canceled individually. A Cancel Message command (0x01) will be immediately sent to all client clusters hosted on an active device for each message. These commands will be sent from the specified endpoint hosting a Messaging server cluster on the local device.

### cancel\_all\_message\_events Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Messaging server cluster. Defaults to the standard SE endpoint (0x5E).

### cancel\_all\_message\_events\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Messaging server cluster.
cluster_id	int	16-bit identifier of the cluster to which the Messaging command was sent.
status	int	Indicates the local success or failure of cancelling the events using standard ZCL status values. A cancel should never fail locally so this field should always contain success (0x00).
device_list	list	List of all devices to which the cancel was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the cancel was sent.

## Messaging Commands - Client

### received\_message\_confirmation (response only)

This RPC response message is generated when a Messaging server cluster hosted on the local device receives a Message Confirmation command (0x01). This command can be sent to confirm a message, or confirm that a message has been canceled.

### received\_message\_confirmation Parameters:

Parameter	Type	Description
source_address	MAC	64-bit extended address of the device hosting the Messaging client.
source_endpoint_id	int	8-bit identifier of the endpoint on the remote device hosting the Messaging client.
cluster_id	int	16-bit identifier of the cluster to which the message command was sent.
record	MessageConfirmationRecord	Describes the status of the event. (See "MessageConfirmationRecord" on page 128)





## received\_message\_event (response only)

This RPC response message is generated when a Messaging client cluster hosted on the local device receives a Display Message command (0x00).

### received\_message\_event Parameters:

Parameter	Type	Description
status	int	Indicates the success or failure of receiving the message event. Can be success (0x00), invalid field (0x85), or duplicate exists (0x8A).
source_address	MAC	64-bit extended address of the device hosting the Messaging server cluster.
source_endpoint_id	int	8-bit identifier of the endpoint on the remote device hosting the Messaging server cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on the gateway that hosts the Messaging client cluster.
cluster_id	int	16-bit identifier of the cluster to which the message command was sent.
record	DisplayMessageEventRecord	Contains information about the received event. (See "DisplayMessageEventRecord" on page 128.)

## received\_message\_cancel\_event (response only)

This RPC response message is generated when a Messaging client cluster hosted on the local device receives a Cancel Message Event command (0x01).

### received\_message\_confirmation Parameters:

Parameter	Type	Description
status	int	Indicates the success or failure of receiving the cancel message event. Can be success (0x00), invalid field (0x85), or not found (0x8B).
source_address	MAC	64-bit extended address of the device hosting the Messaging server cluster.
source_endpoint_id	int	8-bit identifier of the endpoint on the remote device hosting the Messaging server cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on the gateway that hosts the Messaging client cluster.

Parameter	Type	Description
cluster_id	int	16-bit identifier of the cluster to which the message command was sent.
record	CancelMessageEventRecord	Contains information about the received event. (See “CancelMessageEventRecord” on page 128.)

## updated\_active\_message\_events (response only)

The messaging client cluster keeps a list of currently active message events. When this list changes, the client generates an `updated_active_message_events` response. This can occur when an event starts, stops, or is canceled. An empty list of events will be returned when there are no active events.

**Note:** When a Cancel Message Event command that requires a confirmation for the currently active message is received, an `updated_active_message_events` will be generated. The record for the message that was canceled will have an extra parameter “cancel\_message\_control” set to the “message\_control” parameter in the Cancel Message Event.

### updated\_active\_message\_events Parameters:

Parameter	Type	Description
destination_address_id	int	8-bit identifier of the endpoint on the gateway on which the Messaging client cluster exists
cluster_id	int	16-bit identifier of the cluster to which the message command was sent.
record_list	list	List of active message events. List will be empty if there are no active message events. item - DisplayMessageEventRecord (See “DisplayMessageEventRecord” on page 128.)

## get\_last\_message\_event

Get server to resend the most recent message event. Client sends a get last message event command (0x00) to each of the active Messaging servers.

### get\_last\_message\_event Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Messaging client cluster. Defaults to the standard SE endpoint (0x5E).

### get\_last\_message\_event\_response Parameters:

Parameter	Type	Description
status	int	Indicates the success or failure of requesting the last event using ZCL status values. Can be success (0x00), invalid field (0x85), or not found (0x8B).
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Messaging client cluster.
cluster_id	int	16-bit identifier of the cluster from which the message command was sent.
device_list	list	List of all devices to which the cancel was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the cancel was sent.

### confirm\_message

Message events and cancel message events can require confirmation from the Messaging client. When the Messaging client receives a confirm\_message, it will immediately send a Message Confirmation Command (0x01) to all the active Messaging servers. The Message Confirmation Command uses the message ID to specify the message event or cancel message event it is confirming.

### confirm\_message Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Messaging client cluster. Defaults to the standard SE endpoint (0x5E).
record (optional)	MessageConfirmationRecord	Specifies the message ID for the message event or cancel message event being confirmed. Defaults to confirming the currently active message. (See "MessageConfirmationRecord" on page 129.)

### confirm\_message\_response Parameters:

Parameter	Type	Description
status	int	Indicates the local success or failure of confirming the event using standard ZCL status values. Can be success (0x00), invalid field (0x85), or not found (0x8B).
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Messaging client cluster.

Parameter	Type	Description
cluster_id	int	16-bit identifier of the cluster from which the message command was sent.
record (optional)	MessageConfirmationRecord	Specified the message ID for the message event or cancel message event being confirmed. Only include if record is specified in the original confirm_message command (See “MessageConfirmationRecord” on page 129.)
device_list	list	List of all devices to which the confirmation was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the cancel was sent.

## clear\_message\_events

Removes all message events from the gateway’s Messaging client cluster’s list of events. Events are still active on the server and may be resent or requested to be resent by the client using get\_last\_message\_event command (see get\_last\_message\_event on page 106).

### clear\_message\_events Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Messaging client cluster. Defaults to the standard SE endpoint (0x5E).

### clear\_message\_events\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Messaging client cluster.
cluster_id	int	16-bit identifier of the cluster from which the message command was sent.



## ***Price Commands - Common***

### **get\_price\_events**

Returns the gateway's list of valid price events. This command can be used on a price server or client cluster.

#### **get\_price\_events Parameters:**

Parameter	Type	Description
endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Price server cluster. Defaults to the standard SE endpoint (0x5E).
server_or_client	int	Return events from local server (0) or client (1) cluster. If not specified, will search endpoint for Price cluster.

#### **get\_price\_events\_response Parameters:**

Parameter	Type	Description
endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Price cluster.
cluster_id	int	16-bit identifier of the Price cluster.
server_or_client	int	Identified the local cluster the events list is from as a server (0) or client (1) cluster.
events_record_list	int	List of valid events.  <b>item</b> - PublishPriceRecord - Contains information about the event. (See "PublishPriceRecord" on page 128)

## ***Price Commands - Server***

### **create\_price\_event**

Add a price event to the gateway's list of events. A Publish Price command (0x00) will be immediately sent to all Price client clusters hosted on an active device. This command will be sent from the specified endpoint hosting a Price server cluster on the local device.

### create\_price\_event Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Price server cluster. Defaults to the standard SE endpoint (0x5E).
record	PublishPriceRecord	Contains information about the event to be added. (See "PublishPriceRecord" on page 128).

### create\_price\_event\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Price server cluster.
cluster_id	int	16-bit identifier of the cluster to which the Price command was sent.
status	int	Indicates the local success or failure of adding the event using standard ZCL status values. Can be success (0x00), invalid field (0x85) or duplicate exists (0x8A).
record	PublishPriceRecord	Contains information about the event that was added. (See "PublishPriceRecord" on page 128)
device_list	list	List of all devices to which the event was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the event was sent.

### cancel\_all\_price\_events

Removes all price events from the gateway's list of events. The Price cluster does not define cancel commands so Price client clusters cannot be notified of the canceled prices.

### cancel\_all\_price\_events Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Price server cluster. Defaults to the standard SE endpoint (0x5E).

### cancel\_all\_price\_events\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Price server cluster.
status	int	Indicates the local success or failure of cancelling all events using standard ZCL status values. A cancel should never fail locally, so this field should always contain success (0x00).

## **Price Commands - Client**

### **get\_current\_price\_event**

Get server to resend the current price event. Client sends a Get Current Price Event command (0x00) to each of the active price servers.

#### **get\_current\_price\_event Parameters:**

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Price client cluster. Defaults to the standard SE endpoint (0x5E).

#### **get\_current\_price\_event\_response Parameters:**

Parameter	Type	Description
status	int	Indicates the local success or failure of requesting the current price using standard ZCL status values. Can be success (0x00), invalid field (0x85), or not found (0x8B).
endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Price client cluster.
cluster_id	int	16-bit identifier of the cluster from which the Price command was sent.
device_list	list	List of all devices to which the request was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the request was sent.





## get\_scheduled\_price\_events

Get server to resend scheduled price events. Client sends a Get Scheduled Price Events command (0x01) to each of the active Price servers. This request can specify a start time and maximum number of events for the server to send.

### get\_scheduled\_price\_events Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the DRLC client cluster. Defaults to the standard SE endpoint (0x5E).
record (optional)	GetScheduledPrices-Record	Specifies the start time and number of events to get from the DRLC server cluster. Defaults to all events. (See "GetScheduledPricesRecord" on page 130.)

### get\_scheduled\_price\_events\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Price client cluster.
cluster_id	int	16-bit identifier of the cluster from which the Price command was sent.
record (optional)	GetScheduledPrices-Record	Returns the start time and number of events requested from the Price server cluster. Only included in response if specified in the request. (See "GetScheduledPricesRecord" on page 130.)
device_list	list	List of all devices to which the request was sent. Will be empty if status is not success (0x00).  <b>item</b> - MAC - 64-bit extended address of a device to which the request was sent.

## clear\_price\_events

Removes all price events from the gateway's Price client cluster's list of events. Events are still active on the server and may be sent or requested to be resent by the client using the get\_scheduled\_price\_events command.

### clear\_price\_events Parameters:

Parameter	Type	Description
source_endpoint_id (optional)	int	8-bit identifier of the endpoint on the gateway hosting the Price client cluster. Defaults to the standard SE endpoint (0x5E).

### clear\_price\_events\_response Parameters:

Parameter	Type	Description
source_endpoint_id	int	8-bit identifier of the endpoint on the gateway hosting the Price client cluster.
cluster_id	int	16-bit identifier of the Price cluster.

### received\_price\_event (response only)

This RPC response message is generated when a Price cluster hosted on the local device receives a Publish Price Event command (0x00). The event will be added to the Price client.

### received\_price\_event Parameters:

Parameter	Type	Description
status	int	Indicates the success or failure of receiving the price event. Can be success (0x00), invalid field (0x85), or duplicate exists (0x8A).
source_address	MAC	64-bit extended address of the device hosting the Price server cluster.
source_endpoint_id	int	8-bit identifier of the endpoint on the remote device hosting the Price server cluster.
destination_endpoint_id	int	8-bit identifier of the endpoint on the gateway that hosts the Price client cluster.
cluster_id	int	16-bit identifier of the Price cluster.
record	DisplayPriceEvent-Record	Contains information about the received event. (See "DisplayPriceEventRecord" on page 129.)

## Aliasing Commands

Adding, removing and listing aliases are all standard RPC requests (see below). The use of an alias does not follow the RPC format because they can be included anywhere within an RPC request or represent one or more RPC requests. To insert an alias into an RPC request, use the alias name as the tag name and set the type to “alias”. The Python RPC manager will replace the alias with the XML from the add\_alias command. See “Aliases” on page 60 for more information.

### add\_alias

Adds an alias to be used in later RPC requests. The format of this command is as follows:

```
<add_alias>

  <alias_name type="xml">

    XML

  </alias_name>

</add_alias>
```

#### add\_alias Parameters:

Parameter	Type	Description
alias_name	xml	Name of the alias. (Does not have to be “alias_name”.) Name is used to list, remove, and use the alias.

#### add\_alias\_response Parameters:

Parameter	Type	Description
alias_name	string	Name of the alias that was created.

### remove\_alias

Removes a previously defined alias.

The format of this command is as follows:

```
<remove_alias>

  <alias_name/>

</remove_alias>
```

#### remove\_alias Parameters:

Field	Type	Description
alias_name	none	Tag is the name of the alias to be removed.

#### remove\_alias\_response Parameters:

Parameter	Type	Description
alias_name	bool	Tag is the name of the alias that was removed. Set to TRUE if alias was removed. Set to FALSE if alias didn't exist.

## list\_aliases

Lists all aliases that are currently defined. This command takes no parameters.

The format of the response is as follows:

```
<list_aliases_response>

  <alias_list type="dict">

    <alias_name type="xml"><always>look on the bright side of life</always></alias_name>

    <different_alias_name type="xml"><newt>I got better ...</newt></different_alias_name>

  </alias_list>

</list_aliases_response>
```

#### list\_aliases\_response Parameters:

Parameter	Type	Description
alias_list	list	List of aliases of type xml. Each item of the list will use the name of the alias as the tagname.

## RECORD REFERENCE

Records are used on the gateway both internally and as input and output parameters in the RPC Interface. Many of the records correspond to descriptions found in the ZigBee, ZCL, or SE specifications and where appropriate this has been noted. Type, descriptions, defaults, etc for record fields are specific to their use in the RPC Interface. See the appropriate specification for more detailed information.

The referenced specifications are provided by the ZigBee Alliance and can be downloaded from their website at [www.zigbee.org](http://www.zigbee.org).

### ZDO Records

#### ZDODeviceRecord

This record contains the known information about a device on the gateway's ZigBee network. This record is populated and updated as part of discovery explained in "Remote Device Management" on page 42.

Parameter	Type	Description
active	bool	If TRUE, the device is considered active and is responding to the gateway's queries. If FALSE, the device is considered inactive due to failed transmissions. This may be caused by interference, power loss, remote device failures, etc.
addr_extended	MAC	64-bit extended address of the device.
addr_short (optional)	MAC	16-bit short network address of the device. Only included if the device has responded to a ZDO Match_Desc_req.
active_endpoints (optional)	list	List of all endpoints currently active on the device. Only included if the device has responded to a ZDO Active_EP_req.  <b>item</b> - int - 8-bit identifier of an active endpoint.
simple_descriptors (optional)	dict	Dictionary of simple descriptors for active endpoints. Included if the device has responded to a ZDO Simple_Desc_req.  <b>endpoint_0x##</b> - Simple_Desc_rsp - Simple descriptor of the endpoint whose identifier is ##. (See Simple_Desc_rsp on page 119.)
power_descriptor (optional)	Power_Desc_rsp	Power descriptor of the device. Included if the device has responded to a ZDO Power_Desc_req. (See Power_Desc_rsp on page 119.)
node_descriptor (optional)	Node_Desc_rsp	Node descriptor of the device. Included if the device has responded to a ZDO Node_Desc_req. (See Node_Desc_rsp on page 118.)

Parameter	Type	Description
node_type (optional)	int	Identifier of the node type of the device. Included if the device has responded to a ZDO Node_Desc_req.
manufacturer_id (optional)	int	Manufacturer identifier of the device. Included if the device has responded to a ZDO Node_Desc_req.

## Node\_Desc\_rsp

All parameters following nwk\_addr correspond to the Node Descriptor, whose format is given in the [ZigBee Specification](#).

If status is not success (0) then all parameters following nwk\_addr are excluded.

Parameter	Type	Description
status	int	On failure this parameter will be non-zero and the descriptor is not present in the response.
nwk_addr	int	16-bit short network address of the device.
logical_type	int	
complex_desc_avail	int	
user_desc_avail	int	
aps_flags	int	
frequency_band	int	
mac_capability_flags	int	
manufacturer_code	int	
max_buffer_size	int	
max_incoming_transfer_size	int	
server_mask	int	
max_outgoing_transfer_size	int	
descriptor_capability_field	int	

## Power\_Desc\_rsp

All parameters following `nwk_addr` correspond to the Power Descriptor, whose format is given in the [ZigBee Specification](#).

If status is not success (0) then all parameters following `nwk_addr` are excluded.

Parameter	Type	Description
status	int	On failure this parameter will be non-zero and the descriptor is not present in the response.
nwk_addr	int	16-bit short network address of the device.
current_power_mode	int	
available_power_sources	int	
current_power_source	int	
current_power_source_level	int	

## Simple\_Desc\_rsp

All parameters following `nwk_addr` correspond to the Simple Descriptor, whose format is given in the [ZigBee Specification](#).

If status is not success (0) then all parameters following `nwk_addr` are excluded.

Parameter	Type	Description
status	int	On failure this parameter will be non-zero and the descriptor is not present in the response.
nwk_addr	int	16-bit short network address of the device.
endpoint_id	int	8-bit identifier of the endpoint.
profile_id	int	16-bit device identifier of the endpoint.
device_id	int	16-bit device identifier of the endpoint.
device_version	int	4-bit device version of the endpoint
input_clusters	list	List of input clusters for the given endpoint. <b>item</b> - int - 16-bit cluster ID of the input cluster.
output_clusters	list	List of output clusters for the given endpoint. <b>item</b> - int - 16-bit cluster ID of the output cluster.

## ZCL Records

### ReadAttributeRecord

Corresponds to Read Attributes parameters as given in the [ZCL Specification](#).

Parameter	Type	Description
attribute_id	int	Identifier of the attribute that is to be read.

### ReadAttributeStatusRecord

Corresponds to Read attribute status record as documented for Read Attributes Response in the [ZCL Specification](#).

Parameter	Type	Description
status	int	On failure this parameter will be non-zero and the attribute_type and value are not present in the response.
attribute_id	int	Identifier of the attribute that was read.
attribute_type	int	Only included if status is success (0)
value	int	Only included if status is success (0)

### WriteAttributeRecord

Corresponds to Write attribute record as documented for Write Attributes in the [ZCL Specification](#).

Parameter	Type	Description
attribute_id	int	Identifier of the attribute to be written.
attribute_type	int	Data type of the attribute to be written. Defaults to Unknown (0xFF).
value	depends on attribute_type	Value of the attribute to be written. The type of this parameter depends on attribute_type.  For example, if attribute_type specifies Unsigned 16-bit integer (0x21) then value should be an int and if attribute_type specifies Octet string (0x41) then value should be a string.



## WriteAttributeResponseRecord

Corresponds to Write attribute status record as documented for Write Attributes Response in the [ZCL Specification](#)

Parameter	Type	Description
status	int	On failure this parameter will be non-zero.
attribute_id	int	Identifier of the attribute that was written.
attribute_type	int	Copied from the original request parameter.
value	depends on the attribute type	Copied from the original request parameter.

## AttributeReportingConfigurationRecord

Corresponds to Attribute reporting configuration record as documented for Configure Reporting in the [ZCL Specification](#).

The start\_receiving\_reports and stop\_receiving\_reports RPC requests automatically set the direction to transmitting (0x00). Direction should not be included in the record when these commands are used.

Parameter	Type	Description
attribute_id	int	Identifier of the attribute to be configured by this request.
direction	int	If 0x00, then configuration is for transmitting reports. If 0x01, then configuration is for receiving reports. Defaults to 0x00.
attribute_type	int	Include if request is to configure transmitting reports. Defaults to Unknown (0xFF).
min_interval	int	Include if request is to configure transmitting reports. Defaults to 0.
max_interval	int	Include if request is to configure transmitting reports. Defaults to 0.
reportable_change	depends on the attribute type	Include if request is to configure transmitting reports. Defaults to 0.
timeout	int	Include if request is to configure receiving reports. Defaults to 0.

## AttributeReportingConfigurationResponseRecord

Corresponds to Attribute status record as documented for Configure Reporting Response in the [ZCL Specification](#). Parameters other than status and attribute\_id are copied from the original request parameters.

If status is not success (0) then all other parameters except attribute\_id are excluded.

Parameter	Type	Description
status	int	On failure this parameter will be non-zero and only attribute_id is included in the response.
attribute_id	int	Identifier of the attribute that was configured.
direction	int	If 0x00, then configuration was for transmitting reports. If 0x01, then configuration was for receiving reports. Defaults to 0x00.
attribute_type	int	Included if request was to configure transmitting reports. Defaults to Unknown (0xFF).
min_interval	int	Included if request was to configure transmitting reports. Defaults to 0.
max_interval	int	Include if request was to configure transmitting reports. Defaults to 0.
reportable_change	int	Included if request was to configure transmitting reports. Defaults to 0.
timeout	int	Included if request was to configure receiving reports. Defaults to 0.

## AttributeReportRecord

Corresponds to Attribute report as documented for Report Attributes in the [ZCL Specification](#).

Parameter	Type	Description
attribute_id	int	Identifier of the attribute that is being reported.
attribute_type	int	Type of the attribute that is being reported.
value	depends on attribute_type	Value of the attribute that is being reported. The type of this parameter depends on attribute_type.  For example, if attribute_type specifies Unsigned 16-bit integer (0x21) then value should be an int and if attribute_type specifies Octet string (0x41) then value should be a string.

## StopReportingRecord

When passed as a parameter to the stop\_receiving\_reports RPC request this record is used to generate an AttributeReportingConfigurationRecord suitable for stopping reports.

Parameter	Type	Description
attribute_id	int	Identifier of the attribute for which reporting will be stopped.

## StopReportingStatusRecord

A parameter of the stop\_receiving\_reports\_response RPC response.

Parameter	Type	Description
status	int	If success (0) then reporting successfully stopped. If non-zero an error has occurred.
attribute_id	int	Identifier of the attribute for which reporting was stopped.

## ReadReportingConfigurationRecord

Passed as a parameter to the read\_reporting\_configuration RPC request. This record is used to generate an Attribute record as documented for Read Reporting Configuration in the [ZCL Specification](#).

Parameter	Type	Description
attribute_id	int	Identifier of the attribute whose configuration is being requested.

## ReadReportingConfigurationResponseRecord

Corresponds to Attribute reporting configuration record as documented for Configure Reporting Response in the [ZCL Specification](#).

If status is not success (0) then all other parameters except attribute\_id are excluded.

Parameter	Type	Description
status	int	On failure this parameter will be non-zero and only attribute_id is included in the response.
attribute_id	int	Identifier of the attribute whose configuration was requested.

Parameter	Type	Description
direction	int	If 0x00, then configuration is for transmitting reports. If 0x01, then configuration is for receiving reports. Defaults to 0x00.
attribute_type	int	Include if configuration is for transmitting reports. Defaults to Unknown (0xFF).
min_interval	int	Include if configuration is for transmitting reports. Defaults to 0.
max_interval	int	Included if configuration is for transmitting reports. Defaults to 0.
reportable_change	int	Include if configuration is for transmitting reports. Defaults to 0.
timeout	int	Include if configuration is for receiving reports. Defaults to 0.

## AttributeInformationRecord

Corresponds to Attribute information as documented for Discover Attributes Response in the [ZCL Specification](#).

Parameter	Type	Description
attribute_id	int	Identifier of the attribute.
attribute_type	int	Type of the attribute.

## LocalReportingConfigurationRecord

Corresponds to the information stored locally for reporting. This report includes addressing information for the remote device and the reporting configuration.

Parameter	Type	Description
device_address	MAC	64-bit address of the remote device sending or receiving reports
device_endpoint_id	int	8-bit identifier of the remote endpoint which has the reporting cluster
endpoint_id	int	8-bit identifier of the local endpoint which has the reporting cluster
cluster_id	int	16-bit identifier of the local cluster sending or receiving reports
server_or_client	int	Local cluster is a server (0) or client (1) cluster
reporting_direction	int	Gateway is sending reports (0) or receiving reports (1)
pseudo_reporting (optional)	bool	If TRUE, the gateway is reading the attribute and simulating reporting. Only included if reporting_direction is (1).
record	AttributeReporting-Configuration-Record	Reporting configuration values. (See “AttributeReporting-ConfigurationRecord” on page 121.)

## ZCL\_ArrayRecord

Corresponds to the ZCL Array type as documented in the [ZCL Specification](#). For example, this record can be used to define the value field in a ReadAttributeStatusRecord (See “ReadAttributeStatusRecord” on page 119) or WriteAttributeRecord (See “WriteAttributeRecord” on page 119)

Parameter	Type	Description
attribute_type	int	Type of attribute that is in the array.
value	list	List of items in the array.

## SE Records

### *Demand Response / Load Control (DRLC)*

#### **LoadControlEventRecord**

Corresponds to payload of Load Control Event in the [SE Specification](#).

Parameter	Type	Description
issuer_event_id	int	No default.
device_class	int	Defaults to 0x0FFF (all non-reserved device classes).
utility_enrollment_group	int	Defaults to 0x0 (all groups).
start_time	int	Defaults to 0x0 (now).
duration_in_minutes	int	No default.
criticality_level	int	Defaults to 0x1 (lowest non-reserved priority).
cooling_temperature_offset	int	Defaults to 0xFF (denotes not used).
heating_temperature_offset	int	Defaults to 0xFF (denotes not used).
cooling_temperature_set_point	int	Defaults to -0x8000 (denotes not used).
heating_temperature_set_point	int	Defaults to -0x8000 (denotes not used).
average_load_adjustment_percentage	int	Defaults to -0x80 (denotes not used).
duty_cycle	int	Defaults to 0xFF (denotes not used).
event_control	int	Defaults to 0x3 (randomize start and end times).

#### **CancelLoadControlEventRecord**

Corresponds to payload of Cancel Load Control Event in the [SE Specification](#).

Parameter	Type	Description
issuer_event_id	int	No default.
device_class	int	Defaults to 0x0FFF (all non-reserved device classes).
utility_enrollment_group	int	Defaults to 0x0 (all groups).
cancel_control	int	Defaults to 0x1 (use randomization from original event).
effective_time	int	Defaults to 0x0 (now).

## CancelAllLoadControlEventsRecord

Corresponds to payload of Cancel All Load Control Events in the [SE Specification](#).

Parameter	Type	Description
cancel_control	int	Defaults to 0x1 (use randomization from original event).

## ReportEventStatusRecord

Corresponds to payload of Report Event Status in the [SE Specification](#).

Parameter	Type	Description
issuer_event_id	int	No default.
event_status	int	No default.
event_status_time	int	Defaults to current time when record is created.
criticality_level_applied	int	No default.
cooling_temperature_set_point_applied	int	Defaults to 0x8000 (denotes not used).
heating_temperature_set_point_applied	int	Defaults to 0x8000 (denotes not used).
average_load_adjustment_percentage_applied	int	Defaults to 0x80 (denotes not used).
duty_cycle_applied	int	Defaults to 0xFF (denotes not used).
event_control	int	No default.
signature_type	int	Defaults to 0x01 (ECDSA).
signature	base16	42 bytes. Defaults to all zero.

## GetScheduledEventsRecord

Corresponds to payload of Get Scheduled Events in the [SE Specification](#).

Parameter	Type	Description
start_time	int	Defaults to zero (no start time filter)
number_of_events	int	Defaults to zero (no limit on number of events returned)

## Messaging

### DisplayMessageRecord

Corresponds to payload of Display Message in the [SE Specification](#).

Parameter	Type	Description
message_id	int	No default.
message_control	int	Defaults to 0 (normal transmission, low importance, no confirmation required)
cancel_message_control (optional - response only)	int	Only included in the RPC response updated_active_message (See “updated_active_message” on page 105) when a currently active message is canceled, and the Cancel Message command (See “cancel_message” on page 101) has a message_control field that requires confirmation. In this case, the updated_active_message RPC response inserts the message_control parameter from the Cancel Message into the DisplayMessageRecord as the parameter cancel_message_control to signify that the message was canceled, but should still be displayed until the cancel is confirmed.
start_time	int	Defaults to 0x0 (now)
duration_in_minutes	int	No default
message	string	No default

### CancelMessageRecord

Corresponds to payload of Cancel Message in the [SE Specification](#).

Parameter	Type	Description
message_id	int	No default.
message_control	int	Defaults to 0. (normal transmission)



## MessageConfirmationRecord

Corresponds to payload of Message Confirmation in the [SE Specification](#).

Parameter	Type	Description
message_id	int	No default.
confirmation_time	int	Defaults to current time when record is created.

## GetScheduledEventsRecord

Corresponds to payload of Get Scheduled Events in the [SE Specification](#).

Parameter	Type	Description
start_time	int	Defaults to zero (no start time filter)
number_of_events	int	Defaults to zero (no limit on number of events returned)

## Price

### PublishPriceRecord

Corresponds to payload of Publish Price in the [SE Specification](#).

Parameter	Type	Description
provider_id	int	Defaults to 0xFFFFFFFF (denotes invalid, not used).
rate_label	string	Defaults to empty string.
issuer_event_id	int	No default.
utc_time	int	Set to current time when record is transmitted.
currency	int	Defaults to 840 (US dollar).
price_trailing_digit_and_price_tier	int	Defaults to 0x20 (two digits to right of decimal, no related tier).
number_of_price_tiers_and_register_tier	int	Defaults to 0x0 (no price tiers, no related tier).
start_time	int	Defaults to 0x0 (now).
duration_in_minutes	int	No default.
price	int	No default.

Parameter	Type	Description
price_ratio	int	Defaults to 0xFF (denotes not used).
generation_price	int	Defaults to 0xFFFFFFFF (denotes not used).
generation_price_ratio	int	Defaults to 0xFF (denotes not used).
alternate_cost_deliverd	int	Defaults to 0xFFFFFFFF (denotes not used).
alternate_cost_unit	int	Defaults to 0x1 (kg of CO2 per unit measure).
alternate_cost_trailing_digit	int	Defaults to 0xFF.

## GetScheduledPricesRecord

Corresponds to payload of Get Scheduled Prices in the [SE Specification](#).

Parameter	Type	Description
start_time	int	Defaults to zero (no start time filter)
number_of_events	int	Defaults to zero (no limit on number of events returned)



---

## *Appendix B*

---

### **SMART ENERGY CERTIFICATE MANAGEMENT**

All devices that operate in a ZigBee Smart Energy network must have a certificate installed that authenticates the device and allows it to securely join and communicate on the network. A certificate must be issued by the certificate authority (CA). Each certificate is tied to the 64-bit extended address of the device.

Currently there are two types of certificates issued by the CA. Production certificates are intended for use in deployed Smart Energy networks. Only devices which pass stringent testing and are officially certified may be configured with a production certificate. Test certificates may also be issued by the CA and are functionally equivalent to production certificates. However, a device configured with a test certificate will not be able to securely join and communicate on a production network. Test certificates are useful during development and test, for example when communicating with prototype devices that have not yet been certified, but should not be used in an actual deployment.

#### **Certificates on ConnectPort X2 for Smart Energy**

All ConnectPort X2 for Smart Energy gateways are certified and configured by default with a production certificate, allowing them to create or join production networks out of the box. During the course of development a test certificate may be installed. The gateway's original production certificate can be restored at any time by removing the test certificate.

#### **Certificates on Standalone XBee Modules**

Standalone XBee modules, such as the XStick SE, do not implement a specific Smart Energy device and so cannot be certified directly but only as components in larger systems. Due to this fact, standalone XBee modules cannot be preconfigured with production certificates and will not be able to securely communicate with a Smart Energy gateway out of the box.

In order for the standalone XBee to securely communicate on a Smart Energy network a test certificate will need to be obtained from a certificate authority and installed. Additional test certificates may also be necessary for other devices that are configured with production certificates, such as the ConnectPort X2 for Smart Energy.



## Obtaining Test Certificates

Certicom is the only recognized CA for ZigBee Smart Energy. Certicom provides an online form to request test certificates at <http://www.certicom.com/index.php/gencertregister>. Certificates are linked to a specific 64-bit extended address (EUI) which must be provided to Certicom. Before requesting test certificates determine the EUI of all devices which will need test certificates.

## Determining EUI of a ConnectPort X2 for Smart Energy

The EUI of a Smart Energy gateway may be determined in several ways depending on available access.

- Through the web interface, EUI is displayed as Gateway Address under Administration -> System Information -> XBee Network.
- Through the iDigi portal, go to [developer.idigi.com](http://developer.idigi.com). Once signed in to the webpage, click on "XBee Networks" under the Management category in the left pane. Use the Gateway Device ID located on the bottom of the device to look up the Node Address (aka EUI) of the Gateway in the table.
- The RPC interface can be used to retrieve the EUI of the Gateway as well. The easiest way is to send the `get_zigbee_network_status` request (see `get_zigbee_network_status` on page 78). This request will return the EUI as one of the parameters in its response.

```
<get_zigbee_network_status />
```

## Determining EUI of a Standalone XBee Module

The EUI of a standalone XBee module serially attached to your computer can be obtained in the following ways.

- Run the In-Premise Display/Meter Simulator sample on page 30 and open the serial port associated with the XBee. Once opened, the XBee's EUI will be displayed under XBee Settings.
- Run X-CTU (see "Resources" on page 7), and open the serial port associated with the XBee. Once the serial port has been opened and communication established click the Modem Configuration tab and then the Read button under Modem Parameters and Firmware. Once all modem parameters are read the high 32-bits of EUI are listed as SH - Serial Number High and the low 32-bits are listed as SL - Serial Number Low, both under the Addressing subfolder.



## Installing Certificates

Certificates obtained from Certicom should have the following format where ##### will be a long hexadecimal number for each entry.

```
CA Public Key: #####
Device Implicit Cert: #####
Device Private Key: #####
Device Public Key: #####
```

To install certificates onto either the ConnectPort X2 for Smart Energy or a standalone XBee an AT command must be sent to configure the CA Public KEY (ZU), Device Implicit Cert (ZT), and Device Private Key (ZV).

## Installing Certificates on the ConnectPort X2 for Smart Energy

Use the xbee\_AT RPC request to install a certificate. After installing a certificate the network should also be reset. Use the following requests as a template and send in order.

### 1. Configure CA Public Key

```
<xbee_AT>
  <command type="string">ZU</command>
  <value type="base16">#####</value>
</xbee_AT>
```

### 2. Configure Device Implicit Cert

```
<xbee_AT>
  <command type="string">ZT</command>
  <value type="base16">#####</value>
</xbee_AT>
```

### 3. Configure Device Private Key

```
<xbee_AT>
  <command type="string">ZV</command>
  <value type="base16">#####</value>
</xbee_AT>
```



#### 4. Write settings to non-volatile flash

```
<xbee_AT>  
  <command type="string">WR</command>  
</xbee_AT>
```

#### 5. Reset network

```
<xbee_AT>  
  <command type="string">NR</command>  
</xbee_AT>
```

## Installing Certificates on a Standalone XBee Module

The easiest method to install a certificate on a Standalone XBee Module is using the In-Premise Display/Meter Simulator sample on page 30.

If this is not possible X-CTU may also be used to send the necessary AT commands. However, X-CTU does not provide direct support for the certificate AT commands. The command packets must be manually created, entered into the Terminal tab, and sent to the XBee. (For downloading the X-CTU see “Downloads” on page 7 and for accessing the XBee SE Manual see “Online Documentation” on page 7.)

## Reverting/Uninstalling Certificates

Certificates may be removed from either the ConnectPort X2 for Smart Energy or a standalone XBee. Follow instructions in the appropriate section for installing a certificate except with a value of 0 for the ZU (CA Public Key), ZV (Device Implicit Cert), and ZT (Device Private Key) commands.

A ConnectPort X2 for Smart Energy will revert to its original production certificate when this procedure is followed.

## Production Certificates and Modifications

The ConnectPort X2 for Smart Energy underwent official testing through National Technical Systems (NTS) for Smart Energy certification to allow installation of production certificates. The ZigBee Smart Energy Test Specification and other ZigBee documents provide a set of guidelines for what hardware and/or software changes may require recertification. When making any modifications reference these documents and determine if recertification may be required. Digi International may be able to provide support for recertification. Contact your Sales Representative for more details.